AFRL-RI-RS-TR-2014-195

DISTRIBUTED SENSING AND PROCESSING ADAPTIVE COLLABORATION ENVIRONMENT (D-SPACE)

APTIMA, INC.

JULY 2014

FINAL TECHNICAL REPORT

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* JULY 2014	**2. REPORT TYPE** FINAL TECHNICAL REPORT	**3. DATES COVERED** *(From - To)* MAY 2013 – MAY 2014

4. TITLE AND SUBTITLE

DISTRIBUTED SENSING AND PROCESSING ADAPTIVE COLLABORATION ENVIRONMENT (D-SPACE)

5a. CONTRACT NUMBER	FA8750-13-C-0136
5b. GRANT NUMBER	N/A
5c. PROGRAM ELEMENT NUMBER	62788F
5d. PROJECT NUMBER	S2MA
5e. TASK NUMBER	SA
5f. WORK UNIT NUMBER	AP

6. AUTHOR(S)

Georgiy Levchuk, Andres Ortiz, and John-Collonna Romano

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Aptima, Inc.
12 Gill Street, Suite 1400
Woburn, MA 01801

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RISC
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER

AFRL-RI-RS-TR-2014-195

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. PA# 88ABW-2014-3165
Date Cleared: 1 JUL 2014

13. SUPPLEMENTARY NOTES

14. ABSTRACT
With the proliferation of physical military and commercial sensors, and increasing use of open source social media data for situation understanding, traditional stove-piped exploitation solutions no longer achieve required accuracy in a timely manner. Alternatively, the new “cloud” technologies are not appropriate for situation understanding in areas of denial, where computation resources are limited, data not easily moved around, and communication highly constrained and unreliable. D-SPACE is a Distributed Sensing and Processing Adaptive Collaborative Environment designed to perform large-scale data queries in resource-constrained domains, where the data comes from diverse sources, modalities, and contains high degree of interdependencies among the variables. D-SPACE represents such data as large-scale graphs, enabling online graph analytics via inexact graph matching process. D-SPACE distributes graph exploitation among a network of autonomous computational resources, designs the collaboration policy among them, and assures that processing is robust to communication and resource failures. D-SPACE achieves its objectives by highly efficient work balancing among the resources, graph data indexing, message compression and prioritization to reduce communication load, and adaptive roll-back, re-planning, and re-biasing to deal with communication delays and resource failures. When complete, D-SPACE will provide increased autonomy, improved analysis accuracy, and fault-tolerance, while decreasing the time required for data processing.

15. SUBJECT TERMS
D-SPACE, Algorithms, Software prototype, Heterogeneous Computing Resources

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**WILLIAM D. LEWIS**
U	U	U	SAR	44	**19b. TELEPHONE NUMBER (*Include area code*)** **315-330-7707**

TABLE OF CONTENTS

List of Figures

Figure **Page**

List of Tables

1. SUMMARY

This final report contains the results of our work on DSPACE: Distributed Sensing and Processing Adaptive Collaborative Environment. During this 1-year effort, we developed a formal process, algorithms, and software (SW) prototype to enable processing of distributed relational data across multiple autonomous heterogeneous computing resources in environments with limited control, resource failures, and communication bottlenecks. This work was accomplished through five tasks.

- Task 1 developed a formal process to perform data querying via inexact graph matching using the belief propagation algorithm.
- Task 2 developed a distributed collaborative querying model using the belief propagation algorithm. This includes a formal representation of the collaborative data analysis problem, a set of methodologies for query partitioning and task assignment, the communication message structure and the collaborative updates required for the autonomous agents to perform their tasks.
- Task 3 developed a SW prototype of the distributed graph matching framework using the Java Message Service (JMS) API and the Bulk Synchronous Parallel (BSP) processing paradigm.
- Task 4 developed a set of algorithms to improve the solution’s scalability and reliability. This includes adaptive information prioritization based filtering, communication compression algorithms, and strategies for detection and recovery from agent.
- Task 5 conducted validation experiments using randomly generated graph data as well as open source research data to assess the sensitivity of system’s search accuracy to the size and noise in the data, and showcasing benefits and trade-offs of distributed search model using the various scalability strategies.

The aforementioned tasks are described in detail in the following sections.

2. INTRODUCTION

2.1 *Motivation and Summary of Accomplishments*

Amounts of data that need to be collected, examined and shared during Intelligence, Surveillance, and Reconnaissance (ISR) operations are growing fast due to increasing sensor use. In order to manage the resulting data deluge, the U.S. intelligence community is moving away from manual data analysis toward automated processing capabilities, with the focus on two technologies: Cloud-based distributed processing, and autonomous cooperative data exploitation (Figure 1).

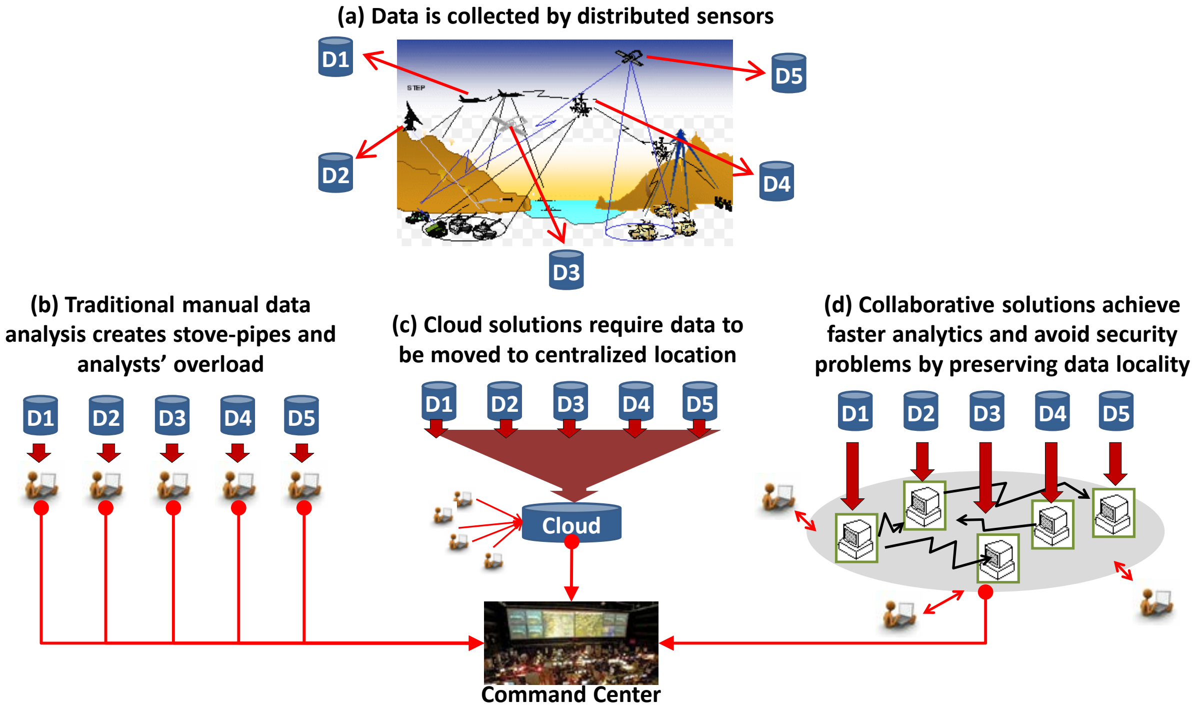

(a) Data is collected by distributed sensors
D1
D2
D3
D4
D5
(b) Traditional manual data analysis creates stove-pipes and analysts' overload
(c) Cloud solutions require data to be moved to centralized location
(d) Collaborative solutions achieve faster analytics and avoid security problems by preserving data locality
Cloud
Command Center

Figure 1: Analysis of data collected by multiple heterogeneous sources (a) is moving away from manual stove-pined processing (b) to Cloud-based (c) or autonomous collaborative solutions (d)

DSPACE focuses on the problem of data exploitation in denied areas, where the control of analytical operations and coordination between distributed data access and computing resources, and even existence of these resources, can be disrupted. Cloud-based technologies are inappropriate for this domain due to weakness of control over computational units and inability to move the data to a central warehouse where it could be indexed, partitioned and analyzed in parallel in a fully controlled environment. Our model provides a solution for autonomous cooperative exploitation of relational data, which is encountered in a variety of applications ranging from geospatial analysis to open source mining.

During this one year effort, we developed a model for processing distributed data across multiple heterogeneous computing resources. Our model exploits the dependencies in the data to provide solutions to both distributed querying and pattern learning. In distributed querying mode, the computing resources are assigned subsets of the query based on high-level information about the data they have access to. This query decomposition is designed to achieve the highest quality of search and optimal balance of computational load between available resources. The resources find local query match estimates and collaborate by exchanging belief messages that efficiently encode how the agents can influence each other's estimates.

In distributed pattern learning mode, the computational resources learn partial correlations between the data they have access to, transfer this knowledge to other agents, and collaboratively learn the patterns within and between their local data subsets.

Our model achieves better-than-linear computational complexity by using several concepts from probabilistic data analysis and belief propagation. First, we minimize the subset of the data being processed at any given time by prioritizing the nodes in the data graph and processing only the subset of highest-priority nodes. This ensures that we perform the least amounts of irrelevant analytical computations. Next, we compress the communication messages by residual (a change from previous value) or absolute value of beliefs, thus minimizing communication between distributed resources. Finally, the computational resources use local data prioritization to incrementally process subsets of data and reduce the computation time of every analysis superstep, resulting in faster production of near-optimal results.

2.2 Background and Functional Requirements

Leading distributed query processing models for sensor networks (Madden et al., 2003; Yao and Gehrke, 2003) try to acquire as much data as possible from the environment while most of that data provides little improvement to approximate answer quality. Hence these models generate the execution costs, in both time and resource utilization, that are orders of magnitude higher than is appropriate for a reasonably reliable answer (Deshpande et al., 2005). While successfully transitioned to distributed cloud systems, these models would not be appropriate for a collaborative analysis domain that has constraints on computation resources and communication bandwidth.

Recently, distributed sensing and data processing has received significant attention in both research and development (Bryant et al., 2008; Dahm, 2010) and acquisition programs. Most existing technologies were developed for raw data processing (e.g., detection of objects in imagery based on networks of cameras; Ding et al., 2012), sensor placement (Mathew, and Surana, 2012), or coordinated planning and scheduling of homogeneous agents (Chen, Levy, and Decker, 2007). These solutions are inadequate for the general collaborative data analysis problem, since it often involves diverse datasets, heterogeneous computation resources, and may contain overlapping or complementary data subsets.

In order to address these challenges and satisfy the functional requirements of this Broad Agency Announcement (BAA) (see Table 2), Aptima developed DSPACE; a system for distributed sensing and processing within an adaptive collaborative environment. Our collaborative distributed data analysis solution leverages the belief propagation algorithm to perform graph mining operations, such as finding inexact matches to queries or model patterns, or learning frequent consistent patterns in the data (Levchuk, Shabarekh, and Furjanic, 2011; Levchuk, Roberts, and Freeman, 2012; Levchuk et al., 2013). We exploit the collaborative nature of belief propagation and probabilistic interpretation of the messages as globally influencing mechanisms for local analytical computations.

Table 1: Functional requirements for distributed collaborative data mining

Requirements in BAA	*What it means*	*D-SPACE solution (Models/Algorithms)*	*Benefits*
"Agents mine knowledge from their own collected data"	**Efficient local data search and inference @ agents**	**Prioritization-based Filtering: local data prioritized / filtered / iteratively processed**	**Reduce time of data analysis**
"Agents represent their findings in a way that can be comprehended by interested parties"	**Generalizable situation representation & collaboration @ agents**	**Belief Propagation Model: local inference / collaborative influence (belief) messages**	**Enable knowledge transfer**
"Agents… identify what information should be propagated and to whom (both individual and shared objectives)"	**Agents send only required/influencing information; do not send their experiences**	**Belief Propagation Model: defines what message are communicated and to whom**	**Obtain optimal solutions in distributed manner and minimize communication needs**
"…agents to complete mission objectives in dynamic and contested environments where … lines of communication can change over time"	**Efficient communication policy and adaptive belief update @ agent**	**Communication Compression, Asynchronous Collaboration, & Agent Failure Recovery: adaptively compress sent messages, belief updates recovery using delay information, agent state monitoring**	**Achieve operation robust to communication constraints**

The idea of using belief propagation as a model for collaborative data analysis is not completely new. Previous research in this area developed the basic building blocks for such a system (Crick, and Pfeffer, 2003; Pfeffer and Tai, 2012; Anker, Dolev, and Hodd, 2008; Chechetka, and Guestrin, 2010; Rogers et al., 2011). Our solution is different from previous work in three ways. First, we use graph matching as a process for collaborative querying, as opposed to full data graph labeling employed in other models. This enables efficient scalability (our algorithms are linear in the size of the data) while maintaining high accuracy of retrieval when data is noisy and queries are ambiguous. For example, even most successful heuristic solutions for inferences on graphs usually require a number of computations of higher-degree polynomials and are capable of only exact pattern search (Aggarwal, Khan, & Yan, 2011; Brocheler, Pugliese, & Subrahmanian, 2010; Khan et al., 2011; Rohloff & Schantz, 2010, 2011). Second, we developed several improvements to collaborative processing. Our derivation of belief updates makes a unique compact formulation which couples efficiently with communication compression and data prioritization operations. We show that communication compression using global belief values works as well or better than the heuristics using belief residuals (Elidan, Mcgraw, and Koller, 2006; Sutton, and McCallum, 2007; Gonzalez, Low, and Guestrin, 2009), and developed exact updates for asynchronous message passing to guarantee optimality under communication delays. Our information prioritization model is equivalent to adaptive partitioning of the data graph, but has a different objective of iterative data processing instead of parallel execution, and thus is different from static graph segmentation and indexing solutions (Brocheler et al., 2010; Malewicz et al., 2010) as well as dynamic partitioning methods (Yang et al., 2012). Finally, although not a part of the above requirements we have developed a unique model for distributed collaborative pattern learning, where the computational resources learn fragments of the patterns in their individual data subsets and collaboratively assemble those patterns into globally coherent structures.

2.3 Research Objectives

The developed DSPACE system aims to achieve the following research objectives:

- Objective 1: formalize a process to perform data querying via inexact graph matching using the belief propagation algorithm.
- Objective 2: develop a distributed collaborative querying model using the belief propagation algorithm.
- Objective 3: develop algorithms to define optimized agent task assignment and collaboration policy and establish a collaboration process using a standard peer-to-peer communication framework.
- Objective 4: develop strategies and algorithms to improve scalability and reliability of the distributed collaborative process
- Objective 5: develop a distributed collaborative pattern learning model.
- Objective 6: conduct experimental validation of accuracy, scalability and adaptability of the solution.

3. METHODS, ASSUMPTIONS, AND PROCEDURES

3.1 *Querying Using Inexact Graph Matching*

Most web search engines provide users the ability to query data using only a set of keywords. Keywords are both a simple way to specify the information needs, and the input structure that requires minimal user interfaces and little to no burden on the users. Recently, answering keyword queries on graph-structured data has emerged as an important research topic (Bhalotia et al., 2002; He et al., 2007). Best search engines have been using the graph knowledge stores behind the scenes to disambiguate the queries, improve the retrieval relevancy and categorizing the search results (Singhal, 2012), and the researchers have been developing approaches to convert the keyword queries into query graphs (Tran et al., 2009). This change of search technologies is rational, because people communicate using stories and not keywords. While construction of graphical queries might be challenging to general users, the expert users can define complex queries with little guidance, and the solutions to support their needs have advanced from SQL query language supporting general database querying to SPARQL[1] query language supporting queries over RDF triple stores. Intelligence analysts in particular have the need to convert their needs and information requirements into multi-attributed query graphs for both manual analysis task distribution and automated search over relational data (Levchuk and Pattipati, 2013). In the following sections we describe our model for distributed collaborative data search, starting with defining the centralized querying problem, and then describing the algorithms to distribute the query among multiple heterogeneous computational resources.

3.1.1 Formal definition of data querying as inexact matching problem

Formally, a complex *query*, also referred to as *model*, is defined as attributed graph $\mathbf{G^M} = (V^\mathbf{M}, E^\mathbf{M}, A^\mathbf{M})$, where $V^\mathbf{M} = \{1, \dots, M\}$ is a set of vertices (representing entities or entity requirements), $E^\mathbf{M} = \{(k, m); k, m \in V^\mathbf{M}\}$ is a set of edges (representing relations between entities), and $A^\mathbf{M} = \{a_{km}^\mathbf{M}\}$ are attributes describing how the entities and their relations *may be observed*, i.e. $a_{kk}^\mathbf{M}$ are attributes (e.g., semantic and syntactic descriptors) of entity k, and $a_{km}^\mathbf{M}$ are attributes of the relation (k, m) between entities k and m. The graph $\mathbf{G^M}$ encoded the knowledge we want to find in the data. Similarly, the knowledge aggregated from multiple sources is defined using attributed *data graph* $\mathbf{G^D} = (V^\mathbf{D}, E^\mathbf{D}, A^\mathbf{D})$, where $V^\mathbf{D} = \{1, \dots, N\}$ $(N \gg M)$ are entity instances or mentions, $E^\mathbf{D}$ are observed relations between these entities, and $A^\mathbf{D} = \{a_{ij}^\mathbf{D}\}$ define actually observed (extracted from text) attributes of entities $a_{ii}^\mathbf{D}$ and relations $a_{ij}^\mathbf{D}$. The *mapping* from the query (model) graph to the data graph is defined as a 0-1 node-to-node assignment matrix $S = \|s_{ki}\|$, where $s_{ki} = 1$ if the node in query k is mapped to the entity mention i, and 0 otherwise (Figure 2).

We can interpret the query graph as consisting of the questions (model graph's nodes) with supported details. The mapping is scored using a conditional posterior probability value (Levchuk, Roberts, and Freeman, 2012; Levchuk, and Pattipati, 2013):

$$P(S|\mathbf{D}, \mathbf{M}) = e^{-\frac{Q(S)}{\eta(\mathbf{G^M})}}. \tag{1}$$

[1] http://www.w3.org/TR/rdf-sparql-query/

Here, $Q(S)$ is a quadratic function of the mismatch between the model graph and mapping-induced subgraph in the data, and the normalization coefficient $\eta(\mathbf{G^M})$ corresponds to the norm of the model:

- Mismatch: $Q(S) = \underbrace{\sum_{ki} s_{ki} C_{ki}}_{node\ mismatch} + \underbrace{\sum_{kmij} s_{ki} s_{mj} C_{kmij}}_{link\ mismatch}$ (2)
- Model norm: $\eta(\mathbf{G^M}) = \underbrace{\sum_k C_{k,null}}_{node\ norm} + \underbrace{\sum_{km} C_{km,null}}_{link\ norm}$

Here, the linear component corresponds to the mismatches between requested (model) attributes and observed (data) attributes – of the nodes $k \in V^{\mathbf{M}}$ and $i \in V^{\mathbf{D}}$, i.e. $C_{ki} \sim mismatch\left(a^{\boldsymbol{D}}_{i,i}, a^{\boldsymbol{M}}_{k,k}\right)$, and the links $(k,m) \in E^{\mathbf{M}}$ and $(i,j) \in E^{\mathbf{D}}$, i.e. $C_{kmij} \sim mismatch\left(a^{\boldsymbol{D}}_{i,j}, a^{\boldsymbol{M}}_{k,m}\right)$, while $C_{k,null}$ and $C_{km,null}$ correspond to the penalties on query nodes and links, e.g. the maximum mismatch or a mismatch to a node/link with no attributes.

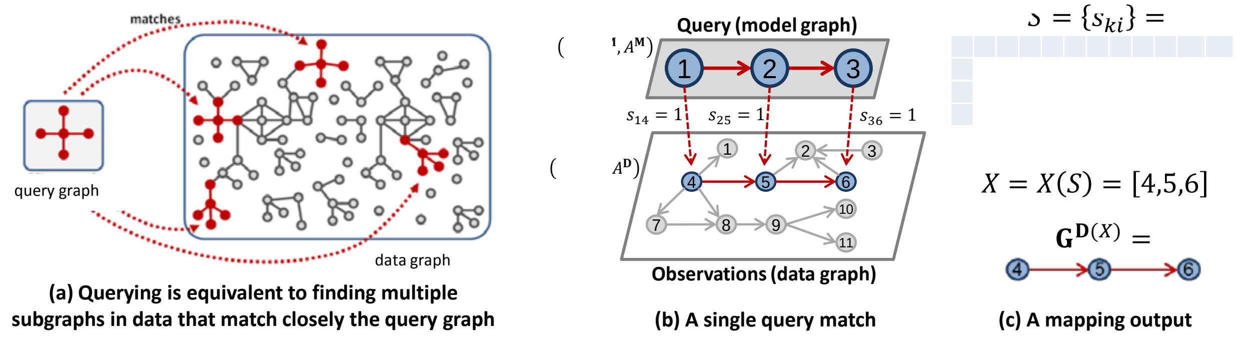


Figure 2: Illustration of graph matching variables

Responses to the query $\mathbf{G^M}$ are represented as tuple:

$$\langle X, \mathbf{G}^{\mathbf{D}(X)} \rangle, \tag{3}$$

where $X = X(S) = \left[x_1, \ldots, x_{|V^{\mathbf{M}}|}\right]$ is a vector of the data variables corresponding to the queries and computed from the mapping matrix S, i.e. $X = \left\{x_k \in V^{\mathbf{D}}, k \in V^{\mathbf{M}}: s_{kx_k} = 1\right\}$; and $\mathbf{G}^{\mathbf{D}(X)} = \left(V^{\mathbf{D}(X)}, E^{\mathbf{D}(X)}, A^{\mathbf{D}(X)}\right)$ is a subgraph in the data graph induced by S, i.e. $V^{\mathbf{D}(X)} = \left\{x_1, \ldots, x_{|V^{\mathbf{M}}|}\right\}$, and $E^{\mathbf{D}(S)} = \{(x_k, x_m) \in E^{\mathbf{D}}: k, m \in V^{\mathbf{M}}\}$. We desire to retrieve responses $\langle X, \mathbf{G}^{\mathbf{D}(X)} \rangle$ which maximize posterior probability from Eq. (1), or equivalently minimize of quadratic mismatch function $Q(S)$ in Eq. (2); in other words, we wish to retrieve the subgraphs of the data that match the query as close as possible.

3.1.2 Solving inexact matching using belief propagation

Due to the factoring of the objective function in Eq. (1), its maximization can be achieved using a Smoothed Loopy Belief Propagation (SLBP) algorithm (Levchuk, Roberts, and Freeman, 2012; Levchuk and Pattipati, 2013). We first incrementally update the estimates of marginal probabilities:

$$b_k(i) = P(\boldsymbol{x}_k = i | \mathbf{D}, \mathbf{M}), \tag{4}$$

for which the joint posterior probability in Eq. (1) is maximized. Marginal probability vector $b_k = [b_k(i), i \in V^{\mathbf{D}}]$ represents a belief about location of query node $k \in V^{\mathbf{M}}$ in the observed dataset.

Original SLBP algorithm obtained the solution to (1) by passing the belief messages in a factor graph (Levchuk, and Pattipati 2013). The *factor graph* is constructed based on factorization of the objective function: it includes the *variable* nodes and *factor* nodes (Figure 3). Variable nodes correspond to query nodes $k \in V^{\mathbf{M}}$; those nodes maintain and update messages $\mu_k = [\mu_k(i), i \in V^{\mathbf{D}}]$ corresponding to the logarithm of marginal beliefs b_k ($\mu_k(i) = \log b_k(i)$), and send these messages to factor nodes. The factor nodes are defined for each link $(k, m) \in E^{\mathbf{M}}$ in the query corresponding to the relationship between query entities; these nodes maintain and update two factor messages: forward and backward. *Forward messages* represent the marginal log-probabilities of matching model link (m, k) to the data link that ends in node j, $f_{(m,k)} = [f_{(m,k)}(j), j \in V^{\mathbf{D}}]$. Forward messages represent an amount of influence that the model node $m \in V^{\mathbf{M}}$ exerts on the decision to map model node $k \in V^{\mathbf{M}}$ to data node $i \in V^{\mathbf{D}}$. *Backward messages* represent the marginal log-probabilities of matching model link (m, k) to the data link that starts in node j, $r_{(m,k)} = [r_{(m,k)}(j), j \in V^{\mathbf{D}}]$. Backward messages represent an amount of influence that the model node $k \in V^{\mathbf{M}}$ exerts on the decision to map model node $m \in V^{\mathbf{M}}$ to data node $j \in V^{\mathbf{D}}$. Essentially, the forward and backward messages can be interpreted as information influencing the search of one query entity based on the other entities that are linked to it.

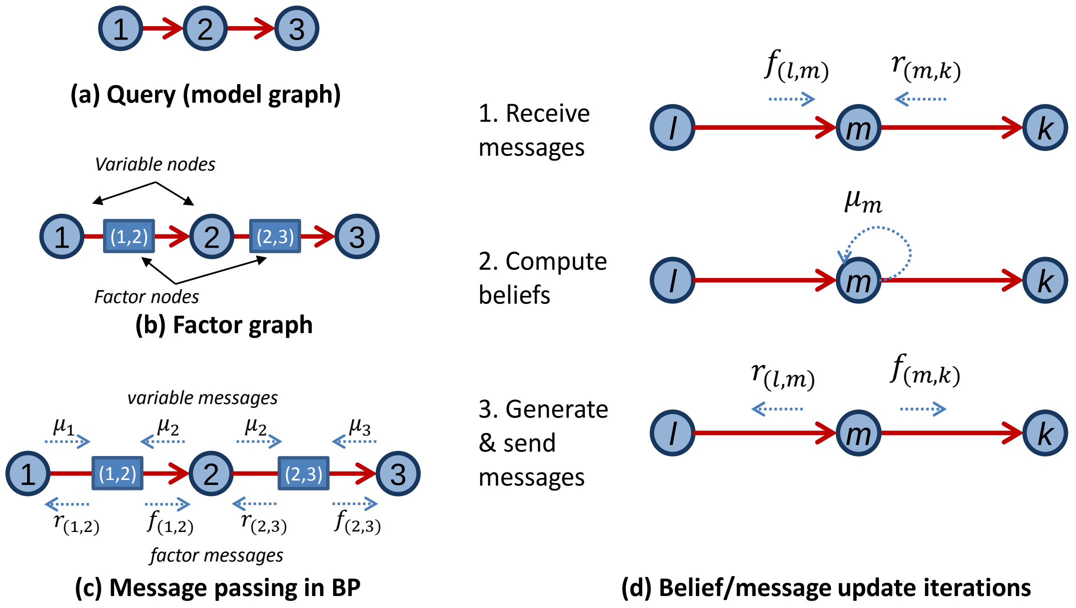


Figure 3: Message passing policy is derived from the structure of the query graph. For the example of query (a), the factor graph (b) contains five variable nodes and four factor nodes. During message passing, variable nodes send their messages to factor nodes, and vice versa. In original formulation, the messages are passed between factor and variable nodes (d)

Formally, the beliefs/messages are updated iteratively in three steps (Figure 3d) using the following equations:

- Updates at variable nodes (local beliefs):

$$\mu_m(i) \propto -C_{mi} + \sum_{l:\,(l,m)\in E^{\mathbf{M}}} f_{(l,m)}(i) + \sum_{l:\,(m,l)\in E^{\mathbf{M}}} r_{(m,l)}(i) \tag{5}$$

- Updates at factor nodes (message generation):

$$f_{(m,k)}(j) \propto \max_{i:\,(i,j)\in E^{\mathbf{D}}}\left(-C_{mk;ij} + \mu_m(i) - r_{(m,k)}(i)\right), \tag{6}$$

$$r_{(m,k)}(j) \propto \max_{i:\,(j,i)\in E^{\mathbf{D}}}\left(-C_{mk;ji} + \mu_k(i) - f_{(m,k)}(i)\right). \tag{7}$$

According to equations (5-7), in-memory single-machine belief propagation requires a total number of message updates and memory storage on the order of $O(\max\{|V_{\mathbf{M}}|, |E_{\mathbf{M}}|\} \times \max\{|V_{\mathbf{D}}|, |E_{\mathbf{D}}|\})$ operations/variables per iteration, while the number of iterations to convergence is on the order of the length of longest path in model network. Thus, since the size of the query graph is usually small, the belief propagation algorithm has *worst-case linear complexity* in the size of the data graph.

In our model, we perform the "smoothing" of the belief updates using reinforcement learning, where the messages $\mu_m(i)$ computed via Eq. (5) at time t are used to incrementally update the smoothed belief estimates $\hat{\mu}_m^t$ based on message estimates $\hat{\mu}_m^{t-1}$ calculated at time $t-1$:

$$\hat{\mu}_m^t = (1-\alpha)\hat{\mu}_m^{t-1} + \alpha\mu_m^t \tag{8}$$

The Eq. (8) attempts to avoid the errors introduced by cycles in the factor graph, which otherwise would create oscillations in the beliefs $\mu_m(i)$, and also provides the effective instrumentation for accounting for message passing delays. This results in using a weighted history of estimates:

$$\hat{\mu}_m^t = (1-\alpha)^t\mu_m^0 + \alpha\sum_{\tau=1}^{t}(1-\alpha)^{t-\tau}\mu_m^\tau \tag{9}$$

When the algorithm terminates at iteration T, we compute the marginal probabilities in Eq. (4) from beliefs:

$$b_k(i) \propto e^{\hat{\mu}_m^T(i)} \tag{10}$$

3.1.3 Generating query output

The query tuples $\langle X, \mathbf{G}^{\mathbf{D}(X)}\rangle$ are generated using probability values $[b_k(i)], \forall k \in V^{\mathbf{M}}, \forall i \in V^{\mathbf{D}}$ using several methods, including K-best maximum weight assignment, marginal probability sampling, conditional sampling, and belief re-evaluation. We describe these methods below.

K-best maximum weight assignment

We approximate the total probability of matching as the product of marginal probabilities:

$$\Pr(X) \sim \prod_{m\in V^{\mathbf{M}}} b_m(x_m) \tag{11}$$

Then the matching can be found as a linear assignment that maximizes the total reward $\sum_{k\in V_{\mathbf{M}}}\hat{\mu}_m^T(x_m)$. To minimize the complexity of solution to assignment problem, we filter the dataset by finding the subset of feasible data points:

$$\hat{V}^{\mathbf{D}} = \{i \in V^{\mathbf{D}}: \max_k b_k(i) \geq threshold\} \tag{12}$$

We then solve a 0-1 assignment problem:

$$\max \sum_{m\in V^{\mathbf{M}}}\sum_{i\in\hat{V}^{\mathbf{D}}}\hat{\mu}_m^T(i)s_{ki} \tag{13}$$

$$s.t.\begin{cases}\sum_{i\in V^{\mathbf{D}}} s_{mi} = 1\\ s_{mi} \in \{0,1\}\end{cases}$$

The set of K ranked "best" solutions to (13) can be obtained using Murty's algorithm (Murty, 1968), which has a computational complexity of $O\left(K\left|\hat{V}^{\mathbf{D}}\right|^{3}\right)$ (Pascoal, Captivo, and Clímaco, 2003). Since the set $\left|\hat{V}^{\mathbf{D}}\right|$ is usually very small, this computational complexity is tractable even for large datasets. The resulting mappings X are then re-ordered using the mismatch cost $Q(X) = \sum_{k\in V^{\mathbf{M}}} C_{kx_k} + \sum_{(k,m)\in E^{\mathbf{M}}} C_{k,m;x_k,x_m}$.

The K-best assignment above may result in the solutions that are far away from optimal, when there are multiple data subgraphs close to the query graph, or subgraph permutations with the same mismatch cost (often the case for homogeneous and/or symmetrical data/query). To avoid this issue, the sampling methods described below can produce improved set of mappings.

Marginal probability sampling

In this method, we generate the mapping $X = \left\{x_1, x_2, \ldots, x_{|V^{\mathbf{M}}|}\right\}$ using binomial distribution, where for each $k \in V^{\mathbf{M}}$ we generate the value $x_k \in V^{\mathbf{D}}$ by drawing samples with probabilities $p_i = \Pr(x_k = i) = b_k(i)$. We then compute a cost of the resulting matching $Q(X)$, and select the set of mappings X that have the smallest score.

Conditional sampling

To derive this method, we draw inspiration from collapsed Gibbs sampling. First, we rewrite the mismatch of mapping X as:

$$Q(X) = \sum_{k\in V^{\mathbf{M}}} \sum_{i\in V^{\mathbf{D}}} C_{ki} s_{ki} + \sum_{(k,m)\in E^{\mathbf{M}}} \sum_{(i,j)\in E^{\mathbf{D}}} C_{k,m;ij} s_{ki} s_{mj} \qquad (14)$$

where $s_{ki} = \begin{cases} 1, x_k = i \\ 0, otherwise \end{cases}$. Accordingly, if a subset of the mapping variables $X^{K-1} = \{x_1, x_2, \ldots, x_{K-1}\}$ is known ($K \leq N$), we can find the expected value of the conditional mismatch as

$$\begin{aligned} E\left(Q(X|X^{K-1})\right) = E_{p(X|X^{K-1})}\left(Q(X)\right) \sim \sum_{k=1}^{K-1} C_{kx_k} + \sum_{k=K}^{N} \sum_{i\in V_{\mathrm{D}}} C_{ki} b_k(i) \\ + \sum_{\substack{(k,m)\in E^{\mathbf{M}}: \\ k<K, m\geq K}} \sum_{j:(x_k,j)\in E^{\mathbf{D}}} C_{k,m;x_k,j} b_m(j) + \sum_{\substack{(k,m)\in E^{\mathbf{M}}: \\ k\geq K, m<K}} \sum_{i:(i,x_m)\in E^{\mathbf{D}}} C_{k,m;i,x_k} b_k(i) \qquad (15) \\ + \sum_{\substack{(k,m)\in E^{\mathbf{M}}: \\ k<K, m<K}} \sum_{(i,j)\in E^{\mathbf{D}}} C_{k,m;x_k,x_m} + \sum_{\substack{(k,m)\in E^{\mathbf{M}}: \\ k\geq K, m\geq K}} \sum_{(i,j)\in E^{\mathbf{D}}} C_{k,m;ij} b_k(i) b_m(j) \end{aligned}$$

Moreover, we observe that conditioning on specific value $x_K = i$ yields:

$$\begin{aligned} E\left(Q(X|X^{K-1}, x_K = i)\right) \sim \sum_{k=1}^{K-1} C_{kx_k} + \sum_{k=K}^{N} \sum_{j\in V^{\mathbf{D}}} C_{ki} b_k(j) \\ + \sum_{k<K:(k,K)\in E_{\mathbf{M}}} C_{k,K;x_k,i} + \sum_{\substack{(k,m)\in E^{\mathbf{M}}: \\ k<K, m>K}} \sum_{j:(x_k,j)\in E^{\mathbf{D}}} C_{k,m;x_k,j} b_m(j) \\ + \sum_{\substack{(k,m)\in E^{\mathbf{M}}: \\ k>K, m<K}} \sum_{i:(i,x_m)\in E^{\mathbf{D}}} C_{k,m;i,x_k} b_k(i) + \sum_{m<K:(K,m)\in E^{\mathbf{M}}} C_{K,m;i,x_k} \qquad (16) \\ + \sum_{\substack{(k,m)\in E^{\mathbf{M}}: \\ k<K, m<K}} \sum_{(i,j)\in E^{\mathbf{D}}} C_{k,m;x_k,x_m} + \sum_{\substack{(k,m)\in E^{\mathbf{M}}: \\ k>K, m>K}} \sum_{(i,j)\in E^{\mathbf{D}}} C_{k,m;ij} b_k(i) b_m(j) \\ + \sum_{m>K:(K,m)\in E^{\mathbf{M}}} \sum_{j:(i,j)\in E^{\mathbf{D}}} C_{K,m;ij} b_m(j) + \sum_{k>K:(k,K)\in E^{\mathbf{M}}} \sum_{j:(j,i)\in E^{\mathbf{D}}} C_{k,K;ji} b_k(j) \end{aligned}$$

Hence, we can compute a conditional sampling probability for variable as follows:

$$p_i^K = \Pr(x_K = i|X^{K-1}) \propto e^{E\left(Q(X|X^{K-1},x_K=i)\right)}$$

$$\sim \exp\left(\begin{array}{c}\sum_{k<K:(k,K)\in E^{\mathbf{M}}} C_{k,K;x_k,i} + \sum_{m<K:(K,m)\in E^{\mathbf{M}}} C_{K,m;i,x_k} \\ + \sum_{k>K:(k,K)\in E^{\mathbf{M}}} \sum_{j:(j,i)\in E^{\mathbf{D}}} C_{k,K;ji} b_k(j) + \sum_{m>K:(K,m)\in E^{\mathbf{M}}} \sum_{j:(i,j)\in E^{\mathbf{D}}} C_{K,m;ij} b_m(j)\end{array}\right) \quad (17)$$

Then, the for each $K \in V^{\mathbf{M}}$ we generate the value $x_K \in V^{\mathbf{D}}$ by drawing samples with probabilities p_i^K computed in Eq. (17).

Belief re-evaluation

The sampling methods described above incorrectly use the global marginal posterior values in the place of local marginal. That is, the posterior probabilities must be re-evaluated during the mapping generation to obtain accurate estimates of $b_K(i|X^{K-1}) = \Pr(x_K = i|X^{K-1}, \mathbf{D}, \mathbf{M})$.

We can achieve this by "continuing" belief propagation algorithm's iterations over a small subset of data points $\hat{V}_{\mathbf{D}}$ found as in Eq. (12) by thresholding the marginal posterior probabilities. The updates will be substituting the mismatch values in Equations (4)-(6) as follows:

- If we map $m \in V^{\mathbf{M}}$ to data $i \in V^{\mathbf{D}}$, i.e. $x_m = i$, then $C_{mj}^{new} \leftarrow \begin{cases} 0, if\ m = k\ and\ j = i \\ \infty, if\ m = k\ or\ j = i \\ C_{mj}, otherwise \end{cases}$

Conducting 1-3 iterations of belief propagation will result in corrections to the $b_k(i)$ values, which can be used in multinomial-based sampling to generate the remaining mapping values.

3.2 Distributed Collaborative Querying Model

Now that we have defined and formalize a procedure for data querying using inexact graph matching we proceed to develop a mechanism that enables querying in a distributed yet collaborative fashion. In the following sections we will provide an overview of the distributed pattern matching problem and our proposed solution, which includes a formal approach to task assignment for the various processing units (agents) and the distributed collaborative formulation of the belief propagation algorithm.

3.2.1 Problem Formulation

We assume that the data graph $\mathbf{G^D}$ is partitioned into subgraphs $\mathbf{G}^{\mathbf{D}(u)}$ such that

$$\mathbf{G^D} = \cup_{u\in U} \mathbf{G}^{\mathbf{D}(u)}, \quad (18)$$

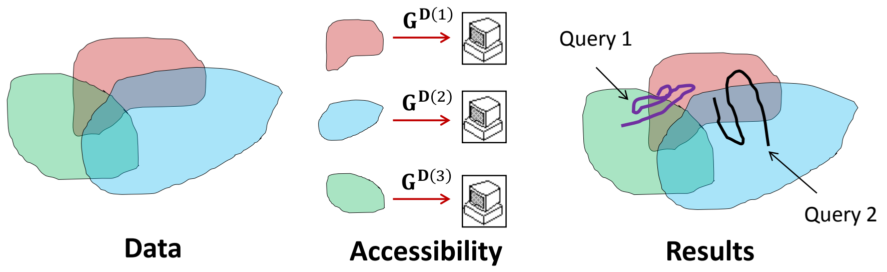

$\mathbf{G}^{\mathbf{D}(1)}$
$\mathbf{G}^{\mathbf{D}(2)}$
$\mathbf{G}^{\mathbf{D}(3)}$
Query 1
Query 2
Data
Accessibility
Results

Figure 4: Data accessibility and query results across partitions

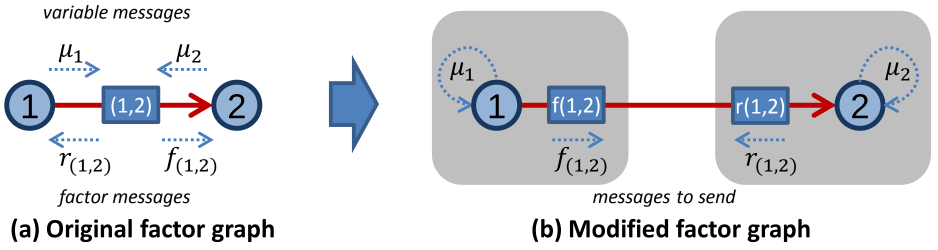
where U is a set of distributed data stores with corresponding computational units, or agents u. Specifically, we define the data graph partition based on the partition of the set of data nodes $V^{\mathbf{D}}$ into subsets $V^{\mathbf{D}(u)}$, where $V^{\mathbf{D}} = \bigcup_{u \in U} V^{\mathbf{D}(u)}$, while the subsets $V^{\mathbf{D}(u)}$ may overlap and the results to queries could be present across overlapping data subsets (Figure 4). Also, we assume that each data subgraphs $\mathbf{G}^{\mathbf{D}(u)} = \left(V^{\mathbf{D}(u)}, E^{\mathbf{D}(u)}, A^{\mathbf{D}(u)}\right)$ contains all the links associated with its nodes, i.e. $E^{\mathbf{D}(u)} = \left\{(i,j) \in E^{\mathbf{D}}: i \in V^{\mathbf{D}(u)} \wedge j \in V^{\mathbf{D}(u)}\right\}$, and the attributes $A^{\mathbf{D}(u)}$ contain a subset of attributes in $A^{\mathbf{D}}$ for corresponding nodes and links, that is $A^{\mathbf{D}(u)} = \left\{a_{ij}^{\mathbf{D}(u)}[k], k \in K(u)\right\}$. For example, the data may be segmented geographically (e.g. when the nodes are areas or tracks), in which case node subsets are non-overlapping: $V^{\mathbf{D}(u)} \cap V^{\mathbf{D}(v)} = \emptyset$ for $u \neq v$. Or the data may be collected over the same areas or entities but by different sensors (e.g., GMTI captures the motion of the objects, while LIDAR captures the characteristics of their 3D shapes), in which case the attribute subsets are non-overlapping: $A^{\mathbf{D}(u)} \cap A^{\mathbf{D}(v)} = \emptyset$ for $u \neq v$.

Distributed graph matching model, as described below, converts the query graph $\mathbf{G}^{\mathbf{M}}$ into the tasks to assign to computation units $u \in U$.

3.2.2 Defining tasks for computation units by restructuring the factor graph

To define the local tasks for computational units $u \in U$, we make the factor graph such as in Figure 5a more compact by splitting each factor node (k,m) into two factors (Figure 5b) that compute forward and backward messages. This allows us to define the query assignment task graph for queries of any structure (Figure 4c shows an example of the task graph for example query in Figure 3a). A *task*, denoted as $\mathrm{T}_m(i,j)$, is defined by aggregating the computations of:

- map-log messages $\mu_m(i)$ from Eq. 5;
- messages from forward factors of all outgoing links $\left\{f_{(m,k)}(j) | \forall k: (m,k) \in E^{\mathbf{M}}\right\}$ from Eq. (6); and
- messages from backward factors of all incoming links $\left\{r_{(k,m)}(j) | \forall k: (k,m) \in E^{\mathbf{M}}\right\}$ from Eq. (7).



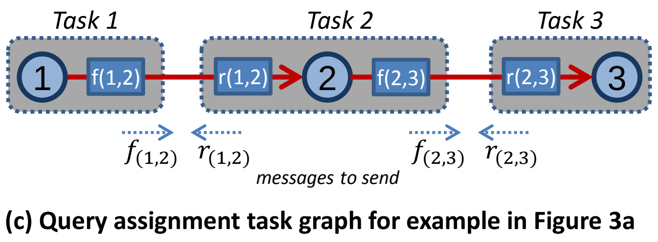
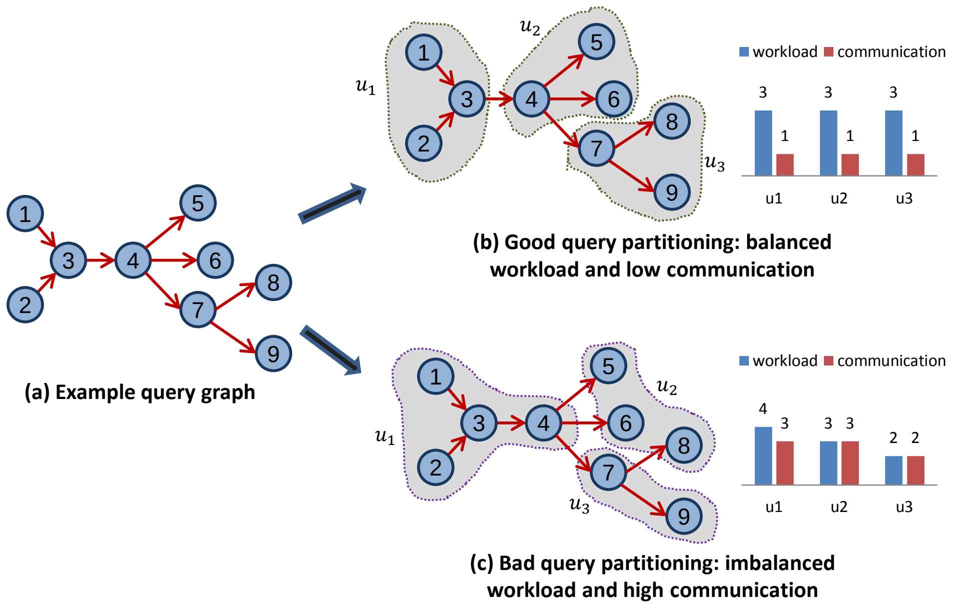


Figure 5: Design of the tasks for distributed computational units

Each task $\mathrm{T}_m(i,j)$ is assigned to one or more computational units, although in most settings an assignment is made to a single unit.

3.2.3 The case of complete data access

In some situations the computational units $u \in U$ may have access to the same data, i.e. $\mathbf{G}^{\mathbf{D}(u)} = \mathbf{G}^{\mathbf{D}}$. In this case, we define and assign the tasks $\mathbf{T}_m = \{\mathrm{T}_m(i,j), i \in V^{\mathbf{D}}, j \in V^{\mathbf{D}}\}$ to compute all the mapping and forward-backward messages. Each such task $\mathbf{T}_m$ corresponds to a single node $m \in V^{\mathbf{M}}$ in the query graph. The assignment of tasks can then be conducted based on the capabilities of computational units. This assignment is equivalent to partitioning (or *cutting*) the query graph into subgraphs, and trading-off the size of the subgraphs and the amount of corresponding communication, which is in the order of number of links separated by corresponding graph cuts (Levchuk, and Pattipati, 2013; Figure 6).



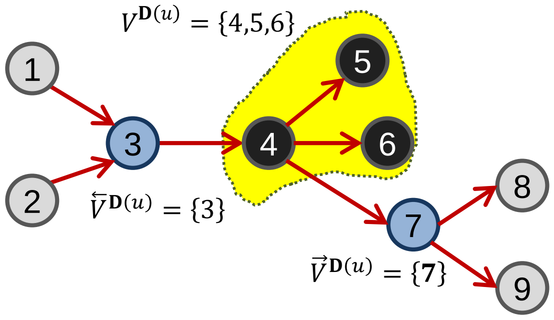
(a) Example query graph
(b) Good query partitioning: balanced workload and low communication
(c) Bad query partitioning: imbalanced workload and high communication
u_1
u_2
u_3
■ workload ■ communication
u1
u2
u3

Figure 6: Examples of query partitioning for distributed assignment

3.2.4 Collaborative distributed belief propagation

Let's assume that computational units $u \in U$ have access to data nodes $V^{\mathbf{D}(u)} \subseteq V^{\mathbf{D}}$ and are assigned to process query nodes $V^{\mathbf{M}(u)} \subseteq V^{\mathbf{M}}$. We make two notations (Figure 7):

- $\vec{V}^{\mathbf{D}(u)}$ is the subsets of data nodes connected with *forward links* from nodes $V^{\mathbf{D}(u)}$, i.e. $\vec{V}^{\mathbf{D}(u)} = \{j \in V^{\mathbf{D}} : \exists i \in V^{\mathbf{D}(u)} \wedge (i,j) \in E^{\mathbf{D}}\}$
- $\overleftarrow{V}^{\mathbf{D}(u)}$ is the subsets of data nodes connected with *backward links* to nodes $V^{\mathbf{D}(u)}$, i.e. $\overleftarrow{V}^{\mathbf{D}(u)} = \{i \in V^{\mathbf{D}} : \exists j \in V^{\mathbf{D}(u)} \wedge (i,j) \in E^{\mathbf{D}}\}$



$V^{\mathbf{D}(u)} = \{4,5,6\}$
1
2
3
4
5
6
7
8
9
$\overleftarrow{V}^{\mathbf{D}(u)} = \{3\}$
$\overrightarrow{V}^{\mathbf{D}(u)} = \{7\}$

Figure 7: Example of subsets of nodes connected to and from data nodes accessed by the computational unit

Then we can write the steps to execute belief propagation in distributed manner:

- **Step 1 – Initialization**: each computational unit $u \in U$ initializes the beliefs and messages:
    - $\forall m \in V^{\mathbf{M}(u)}$, uniformly initialize *beliefs* $\hat{\mu}_m^0 = \{\hat{\mu}_m^0(i), \forall i \in V^{\mathbf{D}(u)}\}$
    - $\forall m \in V^{\mathbf{M}(u)}$, uniformly initialize *external messages*:
        - $\forall l: (m,l) \in E_{\mathbf{M}}$ uniformly initialize forward message $f_{(m,l)}^0 = \left\{f_{(m,l)}^0(j), j \in \overrightarrow{V}^{\mathbf{D}(u)}\right\}$, where $f_{(m,l)}^0(j) = -\ln\left|\overrightarrow{V}^{\mathbf{D}(u)}\right|$
        - $\forall l: (l,m) \in E_{\mathbf{M}}$ uniformly initialize backward message $r_{(l,m)}^0 = \left\{r_{(l,m)}^0(i), i \in \overleftarrow{V}^{\mathbf{D}(u)}\right\}$, where $r_{(l,m)}^0(j) = -\ln\left|\overleftarrow{V}^{\mathbf{D}(u)}\right|$
- **Step 2 – Belief update**: each computational unit $u \in U$ updates beliefs $\hat{\mu}_m^t$ based on its internally computed messages and messages received from other agents, as described in Eq. 5
- **Step 3 – External message generation**: each computational unit $u \in U$ computes the messages $f_{(m,l)}^{t+1}, r_{(l,m)}^{t+1}$ based on belief messages $\hat{\mu}_m^t$ and external messages received from other units $f_{(m,l)}^t, r_{(l,m)}^t$, as described in Eq. (6-7).
- **Step 4 – Communication**: each computational unit $u \in U$ makes decisions to communicate the message or not. It can communicate to agent $v \in U$ the following messages:
    - Forward messages: $\left\{f_{(m,l)}^{t+1}(j), \forall l \in V^{\mathbf{D}(v)}, \forall j \in \overrightarrow{V}^{\mathbf{D}(u)} \cap V^{\mathbf{D}(v)}\right\}$, (19)
    - Backward messages: $\left\{r_{(l,m)}^{t+1}(i), \forall l \in V^{\mathbf{D}(v)}, \forall i \in \overleftarrow{V}^{\mathbf{D}(u)} \cap V^{\mathbf{D}(v)}\right\}$, (20)

The units iterate steps 2-4 until maximum number of iterations is reached or the convergence criteria are met. These four steps define *a collaborative distributed data analysis* because the units influence each other belief computations using communicated forward and backward messages. These messages encode the influence that one unit tries to exert on the beliefs of another unit.

After the belief propagation process is completed at some time T, the units send their beliefs $\hat{\mu}_m^T$ to a unit responsible for generating a final solution. It proceeds by combining received messages, computing marginal mapping probabilities using Eq. (10), and generating query outputs as described in Section 3.1.3.

3.3 Scalability Improvements to Collaborative Query Model

Complexity of collaborative querying using inexact graph matching can be further reduced by limiting local computations and reducing communication requirements. Number of local computations can be reduced at every iteration if the computational units process a subset of accessible data nodes. Communication requirements can be reduced by compressing the number of forward/backward messages. We provide the details of these improvements in the following sections.

3.3.1 Data Prioritization

The computational complexity of the original distributed belief propagation algorithm, computed as the number of summation and maximization operations, for both belief updates and message generation, per each model node and link, is equal to $O\left(\max\left(\left|V^{\mathbf{D}(u)}\right|, \left|E^{\mathbf{D}(u)}\right|\right)\right)$. While this means complexity is linear in the size of the data, many of the computations are unnecessary since the number of relevant nodes is small. We can reduce this complexity by orders of magnitude if we sort the data nodes using a priority measure and select a subset of nodes $V^{\mathbf{D}(u,t)}$ with highest priorities. We define a priority of data node $i \in V^{\mathbf{D}(u)}$ using normalized belief estimates maximized over model nodes:

$$\rho^{(t)}(i) = \max_m \mu_m^{(t)}(i) \tag{21}$$

The notation $\mu_m^{(t)}$ represents the normalized belief estimates:

$$\mu_m^{(t)}(i) = \hat{\mu}_m^t(i) - \log \textstyle\sum_i e^{\hat{\mu}_m^t(i)} \tag{22}$$

These messages are initialized together with belief messages: $\mu_m^{(0)}(i) = -C_{mi} - \log \sum_i e^{-C_{mi}}$ (we change initialization from uniform to incorporating the mismatch values), and accordingly priorities are initialized as $\rho^{(0)}(i) = \max_m\{-C_{mi} - \log \sum_i e^{-C_{mi}}\}$. Normalization is essential for correct updates. The higher the priority $\rho^{(t)}(i)$ is, the more probable at time t it is that the data node i matches at least one of the query nodes asigned to the agent.

After data node priorities are initialized, we select the subset of nodes with highest priorities $V^{\mathbf{D}(u,t)}$ and construct the heap to store the priorities of remaining nodes $V^{\mathbf{D}(u)} \backslash V^{\mathbf{D}(u,t)}$. At every iteration, as the new marginal posterior probability is computed $\hat{\mu}_m^{t+1}(i)$ for node $i \in V^{\mathbf{D}(u,t)}$, we update the priority $\rho^{(t+1)}(i)$ according to Eq. (21) and compare with the root (best value) of the heap. If the updated priority is smaller, the node $i \in V^{\mathbf{D}(u,t)}$ is removed from $V^{\mathbf{D}(u,t)}$ and its value $\rho^{(t+1)}(i)$ added to a heap, while the root of the heap j is added to the "relevant node set" $V^{\mathbf{D}(u,t)}$. We can formally write the update of the relevant set as:

$$\begin{aligned}
&V^{\mathbf{D}(u,t)}[0] = V^{\mathbf{D}(u,t)} \\
&V^{\mathbf{D}(u,t)}[k+1] = \begin{cases} \left\{V^{\mathbf{D}(u,t)}[k] \backslash i_k\right\} \cup j_k, if\ \rho^{(t)}(i_k) < \rho^{(t)}(j_k) \\ V^{\mathbf{D}(u,t)}[k], otherwise \end{cases} \\
&V^{\mathbf{D}(u,t+1)} = V^{\mathbf{D}(u,t)}[|\mathbf{D}(u,t)|]
\end{aligned} \tag{23}$$

where j_k is a root of the heap structure for nodes $\{j \in V^{\mathbf{D}(u)} \backslash V^{\mathbf{D}(u,t)}[k]\}$. Even considering added computational complexity of heap updates, the reduction from processing set $V^{\mathbf{D}(u)}$ to a subset of nodes $V^{\mathbf{D}(u,t)}$ provides equivalent reduction in computational complexity of updates in equations (5-7).

3.3.2 Communication Compression

The number of values in communicated forward and backward messages from Eq. (19-20) are equal in the original formulation to $|\vec{V}^{\mathbf{D}(u)}|$ and $|\overleftarrow{V}^{\mathbf{D}(u)}|$, respectively, can be large, especially when all computational units have access to the same data. However, not all of these messages are relevant. For example, Figure 8 shows how values of forward belief messages change over time: the support of relevant values is gradually reducing. Accordingly, we first use a simple filtering by communicating only a subset of forward/backward messages with largest values. The messages that are not received are replaced with low bound values. Second, we reduced the number of communicated messages by quantizing the message value vector. Quantization uses the hierarchical clustering of the data nodes (Figure 9b) and makes the decision to send a single value for the group of nodes from a cluster, or all individual values. These decisions are generated in a bottom-up manner using the metric of entropy at the cluster level (Figure 9c). Using the definition of belief messages, for a cluster c and a set of data nodes in this cluster $V_c^{\mathbf{D}}$, we compute the entropy for forward and backward messages as $E_{(m,k)}^f = -\sum_{j \in V_c^{\mathbf{D}}} f_{(m,k)}^t(j) \cdot e^{f_{(m,k)}^t(j)}$ and $E_{(m,k)}^r = -\sum_{j \in V_c^{\mathbf{D}}} r_{(m,k)}^t(j) \cdot e^{r_{(m,k)}^t(j)}$, respectively. Consequently, a compressed vector represents a quantization of the belief message.

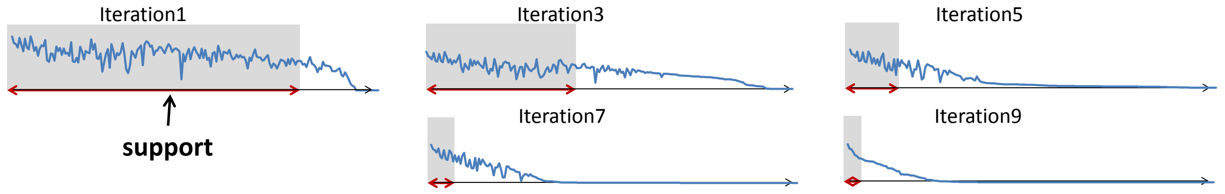
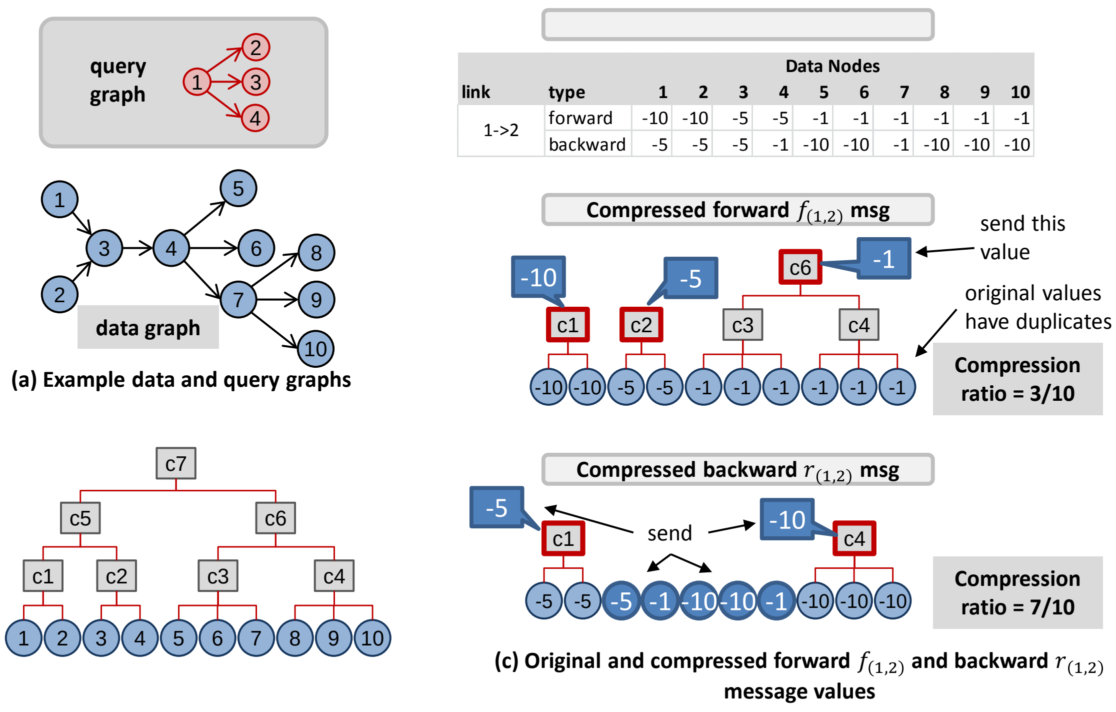


Figure 8: Example of changes in the values of forward messages over iterations

3.3.3 Complexity Reduction Using Question-Answer Process

One of the challenges in collaborative data querying is the need for computational resources to communicate significant amounts of information to each other. While this information can be filtered and compressed, there will still be a significant amount of irrelevant information that one unit may send to another. To avoid this problem, as well as allow units to better discriminate seemingly equivalent entities/nodes in their local data, units can employ a *question-answer process* that will allow them to *request* information of interest rather than push this information to other units. Figure 10 shows an example of this process in the case of a single model node assigned to each of two units. Each unit has computed a current estimate of node-to-node associations between model and data graphs. Then the sets of high-scoring mappings will be limited, e.g., data nodes {2,5} and {5,8} for agents u and v respectively (dotted red arrows in Figure 10). Then, instead of communicating all computed forward/backward messages, each agent asks a question about their own nodes of interest, and will receive only the messages corresponding to these nodes. This will result in significant reduction in amount of communication between the agents (4x reduction in the example depicted in Figure 10, where the communication over dashed directions is not needed).


query graph
data graph
(a) Example data and query graphs
Data Nodes
Compressed forward $f_{(1,2)}$ msg
send this value
original values have duplicates
Compression ratio = 3/10
Compressed backward $r_{(1,2)}$ msg
send
Compression ratio = 7/10
(c) Original and compressed forward $f_{(1,2)}$ and backward $r_{(1,2)}$ message values

link	type	1	2	3	4	5	6	7	8	9	10
1->2	forward	-10	-10	-5	-5	-1	-1	-1	-1	-1	-1
	backward	-5	-5	-5	-1	-10	-10	-1	-10	-10	-10

Figure 9: Example of belief message compression

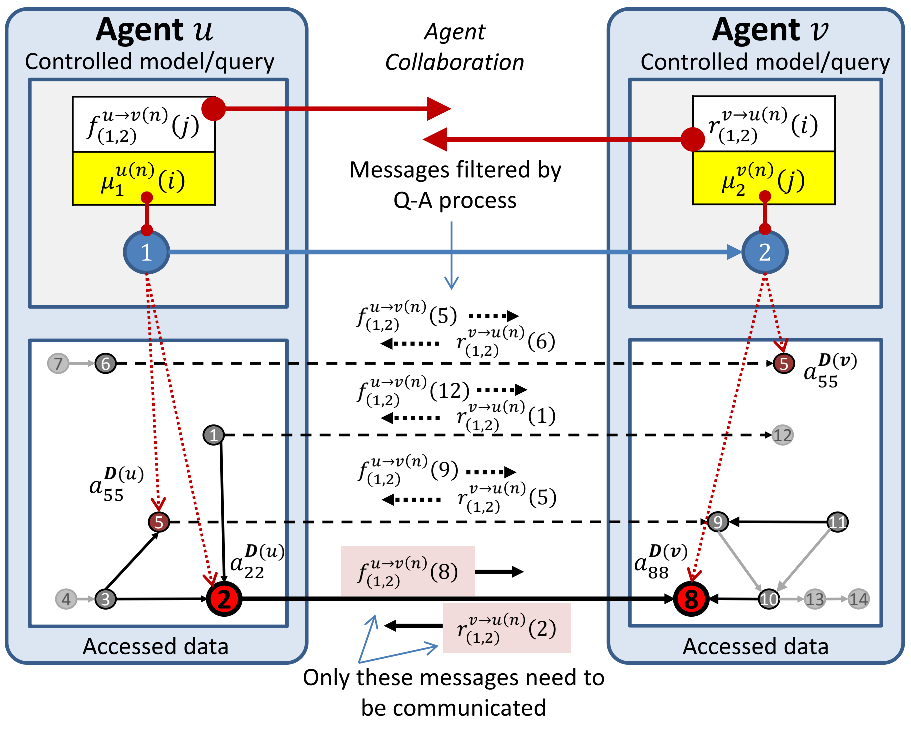

Agent u
Controlled model/query
Agent Collaboration
Agent v
Controlled model/query
$f_{(1,2)}^{u\to v(n)}(j)$
$\mu_1^{u(n)}(i)$
$r_{(1,2)}^{v\to u(n)}(i)$
$\mu_2^{v(n)}(j)$
Messages filtered by Q-A process
$f_{(1,2)}^{u\to v(n)}(5)$
$r_{(1,2)}^{v\to u(n)}(6)$
$f_{(1,2)}^{u\to v(n)}(12)$
$r_{(1,2)}^{v\to u(n)}(1)$
$f_{(1,2)}^{u\to v(n)}(9)$
$r_{(1,2)}^{v\to u(n)}(5)$
$f_{(1,2)}^{u\to v(n)}(8)$
$r_{(1,2)}^{v\to u(n)}(2)$
$a_{55}^{D(u)}$
$a_{22}^{D(u)}$
$a_{55}^{D(v)}$
$a_{88}^{D(v)}$
Accessed data
Accessed data
Only these messages need to be communicated

Figure 10: Question-answer process can reduce number of communicated belief messages

3.4 Collaborative Pattern Learning

Oftentimes structured queries may be weakly designed due to the lack of understanding of phenomena present in the dataset. Also, users may be interested in finding normal (frequent) and abnormal patterns occurring in the dataset under investigation. This can be achieved by distributed collaborative pattern learning.

3.4.1 Types of Learning

Depending on what observed data is accessible by which computational units, we can distinguish two classes of learning: *compositional* learning, and *unified* learning (Figure 11). In compositional learning the units locally learn frequent patterns of node connections in the data they have access to, and then combine these patterns into higher-level "composite" patterns constructed by stitching learned pattern graphlets (Figure 11a). In unified learning, units may learn the patterns of the same type, and then fuse these together to come up with more general frequent graphs (Figure 11b). In this work, we focused on compositional learning, since in many of the problems we encounter, computational units have access to distinct sensor modalities. Accordingly, using a compositional approach to pattern learning would be more appropriate due to existence of different connections across diverse sensor modalities.

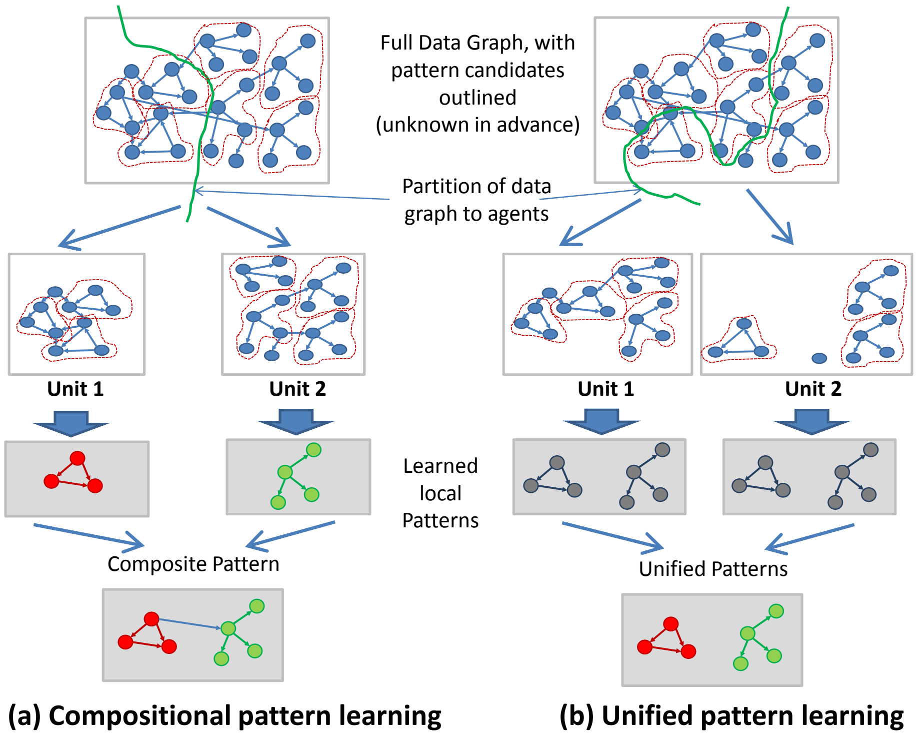

Full Data Graph, with pattern candidates outlined (unknown in advance)
Partition of data graph to agents
Unit 1
Unit 2
Unit 1
Unit 2
Learned local Patterns
Composite Pattern
Unified Patterns
(a) Compositional pattern learning
(b) Unified pattern learning

Figure 11: Different types of pattern learning

3.4.2 Learning Patterns from Graph Instances

In previous work, we developed a model for learning frequent graph patterns in the relational data (Levchuk, Roberts, and Freeman, 2012). Our pattern learning algorithm was based on segmenting the graph into the subgraphs based on topological matching. The training corpus was created by clustering these subgraphs based on the attributed graph similarity, creating the "instances" from which the pattern is learned using a variant of *expectation maximization* (EM) algorithm (Dempster, Laird, and Rubin, 1977) that performs iterative update of model-to-data mappings and model's attributes. For this work, we modified the learning algorithm to avoid computing exact mappings at every iteration, and develop a distributed version of the learning model.

Formally, the attributes of the frequent graph pattern $A^{\mathbf{M}} = \left\{a_{kn}^{\mathbf{M}}\right\}$ are learned from a set of T observations of subgraphs $\mathbf{G}^{\mathbf{D}}(t) = \left(V^{\mathbf{D}}(t), E^{\mathbf{D}}(t), A^{\mathbf{D}}(t)\right), t = 1, \dots, T$ with attributes $A^{\mathbf{D}}(t) = \left\{a_{ij}^{\mathbf{D}}(t)\right\}$, which we call a *learning corpus*. We estimate the attributes of the pattern $\mathbf{M}$ that most likely generated such data by maximizing posterior probability of the pattern given its observed instances (in the equations below we use notations of attributes A, instead of graphs $\mathbf{G}$, for both model and data under the probability functions without loss of generality and for clarity of the expositions):

$$A^{\mathbf{M}} = \arg\max_A P(A|A^{\mathbf{D}}(t), t = 1, \dots, T). \tag{24}$$

We use a variant of the EM algorithm that treats mappings as unobserved variables and iteratively updates pattern attribute parameters. We modified EM from working with the likelihood function to using a posterior distribution. This was dictated by the need to avoid dealing with the large number of components of the objective function in the expectation step. The algorithm proceeds in two iterative steps:

- In the *expectation step*, also called mapping updating step, we find components of the distribution $p_t = P\left(S^t \,|A_n^{\mathbf{M}}, A^{\mathbf{D}}(t)\right)$, so that the expected value of log-posterior function can be calculated: $Q(A|A_n^{\mathbf{M}}) = E_{S^t|A^D(t), A_n^{\mathbf{M}}}\left[\log P\left(A, S^t \middle| A^{\mathbf{D}}(t)\right)\right]$.
- In the *maximization step*, also referred to as parameter updating step, we compute new graph pattern attributes $A_{n+1}^{\mathbf{M}} = \left\{a_{km}^{\mathbf{M}(n+1)}\right\}$ to maximize conditional expected log posterior $A_{n+1}^{\mathbf{M}} = \arg\max Q(A|A_n^{\mathbf{M}})$.

The expectation step is equivalent to conducting iterations via Eq. (5)-(7), to obtain estimates of node and link probabilities, i.e., beliefs and forward-backward messages values $\mu_k^{(n,t)}(i), r_{(k,m)}^{(n,t)}(i), f_{(k,m)}^{(n,t)}(j)$, for mapping at step n a model graph with current attributes $A_n^{\mathbf{M}} = \left\{a_{km}^{\mathbf{M}(n)}\right\}$ to the data graph with attributes $A^{\mathbf{D}(t)}$:

$$\mu_m^{(n,t)}(i) \propto -\left\|a_{mm}^{\mathbf{M}(n)} - a_{ii}^{\mathbf{D}(t)}\right\| + \textstyle\sum_{l:\,(l,m)\in E^{\mathbf{M}}} f_{(l,m)}^{(n,t)}(i) + \sum_{l:\,(m,l)\in E^{\mathbf{M}}} r_{(m,l)}^{(n,t)}(i) \tag{25}$$

$$f_{(m,k)}^{(n,t)}(j) \propto \max_{i:\,(i,j)\in E^{\mathbf{D}}}\left(-\left\|a_{mk}^{\mathbf{M}(n)} - a_{ij}^{\mathbf{D}(t)}\right\| + \mu_m^{(n,t)}(i) - r_{(m,k)}^{(n,t)}(i)\right) \tag{26}$$

$$r_{(m,k)}^{(n,t)}(j) \propto \max_{i:\,(j,i)\in E^{\mathbf{D}}}\left(-\left\|a_{mk}^{\mathbf{M}(n)} - a_{ji}^{\mathbf{D}(t)}\right\| + \mu_m^{(n,t)}(i) - f_{(m,k)}^{(n,t)}(i)\right) \tag{27}$$

To complete maximization step, for the case of Gaussian error of attribute generation, we simplify log-posterior probability computation:

$$\log P\left(A, S^t \middle| A^{\mathbf{D}}(t)\right) \sim \textstyle\sum_{k\in V^{\mathbf{M}}, i\in V^{\mathbf{D}(t)}}\left\|a_{kk} - a_{ii}^{\mathbf{D}(t)}\right\| s_{ki} + \sum_{(k,m)\in E^{\mathbf{M}}, i,j\in V^{\mathbf{D}(t)}}\left\|a_{km} - a_{ij}^{\mathbf{D}(t)}\right\| s_{ki}s_{mj}$$

Next, the expected value of the sum is equal to the sum of expected values of components, hence we obtain:

$$Q(A|A_n^{\mathbf{M}}) = E_{S^t|A^D(t), A_n^{\mathbf{M}}}\left[\log P\left(A, S^t \middle| A^{\mathbf{D}}(t)\right)\right]$$

$$\sim \sum_{k\in V^{\mathbf{M}}, i\in V^{\mathbf{D}(t)}} \left\| a_{kk} - a_{ii}^{\mathbf{D}(t)} \right\| \Pr\left(s_{ki} = 1 \middle| A^{\mathbf{M}}, A^{\mathbf{D}(t)}\right) + \sum_{(k,m)\in E^{\mathbf{M}}, i,j\in V^{\mathbf{D}(t)}} \left\| a_{km} - a_{ij}^{\mathbf{D}(t)} \right\| \Pr\left(s_{ki}s_{mj} = 1 \middle| A^{\mathbf{M}}, A^{\mathbf{D}(t)}\right)$$

$$= \sum_{k\in V^{\mathbf{M}}, i\in V^{\mathbf{D}(t)}} \left\| a_{kk} - a_{ii}^{\mathbf{D}(t)} \right\| e^{\mu_k^{(n,t)}(i)} + \sum_{(k,m)\in E^{\mathbf{M}}, (i,j)\in E^{\mathbf{D}(t)}} \left\| a_{km} - a_{ij}^{\mathbf{D}(t)} \right\| e^{r_{(k,m)}^{(n,t)}(i) + f_{(k,m)}^{(n,t)}(j)}$$

Then, the maximization step results in weighted averaging over mapped attributes of nodes and links:

- Node attributes update: $a_{kk}^{\mathbf{M}(n+1)} = \sum_t \sum_{i\in V^{\mathbf{D}(t)}} a_{ii}^{\mathbf{D}(t)} \cdot e^{\mu_k^{(n,t)}(i)}$; (28)
- Link attributes update: $a_{km}^{\mathbf{M}(n+1)} = \sum_t \sum_{(i,j)\in E^{\mathbf{D}(t)}} a_{ij}^{\mathbf{D}(t)} \cdot e^{r_{(k,m)}^{(n,t)}(i) + f_{(k,m)}^{(n,t)}(j)}$ (29)

Instead of computing the exact estimates in the expectation step, we only need their approximate values; given the incremental changes to the learned pattern, this assumption is warranted. Hence, we conduct few (2 to 3) iterations of the SLBP algorithm to obtain values $\mu_k^{(n,t)}(i), r_{(k,m)}^{(n,t)}(i), f_{(k,m)}^{(n,t)}(j)$ initialized with previous values $\mu_k^{(n-1,t)}(i), r_{(k,m)}^{(n-1,t)}(i)$ and $f_{(k,m)}^{(n-1,t)}(j)$. Thus, the complexity of attributed graph learning remains similar to the original SLBP, with only the increase in the number of iterations and introduction of the computations from equations (28),(29), which are of similar complexity as SLBP iterations. However, this increase may be significant *when the initial estimate of the model graph and/or model-data mapping variables are inaccurate*.

It was shown in (Yedida, Freeman, and Weiss, 2004) that the belief propagation (BP) algorithm minimizes the "free energy" function, which is the difference between the mismatch between mapped attributes of model and data graphs, and the entropy of the marginal mapping distribution. This means that the BP algorithm tries to trade-off finding multiple matches to the model graph in the data, and assuring these are "good" matches. This is exactly the property we desire in learning the graph patterns from data. For the case of "mega-example" training data, i.e., a single large data graph without segmentation into "graph instances", the equations above will still hold if the symbol notation t is dropped. Accordingly, without loss of generality, we remove this notation in the following section.

3.4.3 Collaborative Distributed Learning Model

The learning algorithm described in the previous section represents centralized pattern learning that can be executed by a single unit that has access to all nodes, links, and attributes of the data graph. We extended this algorithm to a distributed case, where the unit u with access to data subgraph $\mathbf{G}^{\mathbf{D}(u)} \subseteq \mathbf{G}^{\mathbf{D}}$ is maintaining and updating beliefs $\mu_k^{u(n)}(i), r_{(k,m)}^{u(n)}(i), f_{(k,m)}^{u(n)}(j)$. In the M-step, the following updates are executed by unit u:

- Update model attributes for nodes $k \in V^{\mathbf{M}(u)}$ assigned to unit u:

$$a_{kk}^{\mathbf{M}(u)} = \sum_{i\in V^{\mathbf{D}(u)}} a_{ii}^{\mathbf{D}} \cdot e^{\mu_k^{u(n)}(i)}$$

- Update model attributes for links $(k,m) \in E^{\mathbf{M}(u)}$ between model nodes assigned to u:

$$a_{km}^{\mathbf{M}(u)} = \sum_{(i,j)\in E^{\mathbf{D}(u)}} a_{ij}^{\mathbf{D}(u)} \cdot e^{r_{(k,m)}^{u(n)}(i)+f_{(k,m)}^{u(n)}(j)}$$

- Update model attributes for links $(k,m) \in E^{\mathbf{M}(u,v)}$ between model nodes outgoing from agent u to agent v:

$$a_{km}^{\mathbf{M}(u,v)} = \sum_{\substack{(i,j)\in E^{\mathbf{D}}:\\ i\in V^{\mathbf{D}(u)},j\in V^{\mathbf{D}(v)}}} a_{ij}^{\mathbf{D}(u,v)} \cdot e^{\underbrace{r_{(k,m)}^{v\to u(n)}(i)}_{received\ from\ unit\ v} + \underbrace{f_{(k,m)}^{u\to v(n)}(j)}_{computed\ at\ unit\ u}}$$

In addition, the unit u will send unit v learned attributes $a_{km}^{\mathbf{M}(u,v)}$ of the link $(k,m) \in E^{\mathbf{M}(u,v)}$ between model nodes $k \in V^{\mathbf{M}(u)}$ and $m \in V^{\mathbf{M}(v)}$ that originated in u. Schematically, the set of variables to perform pattern learning is depicted in the example in Figure 12. As we can see, pattern learning differs from pattern matching by the need to update model graph attributes (which are fixed in the matching process) based on the same beliefs and messages computed and communicated in collaborative matching.

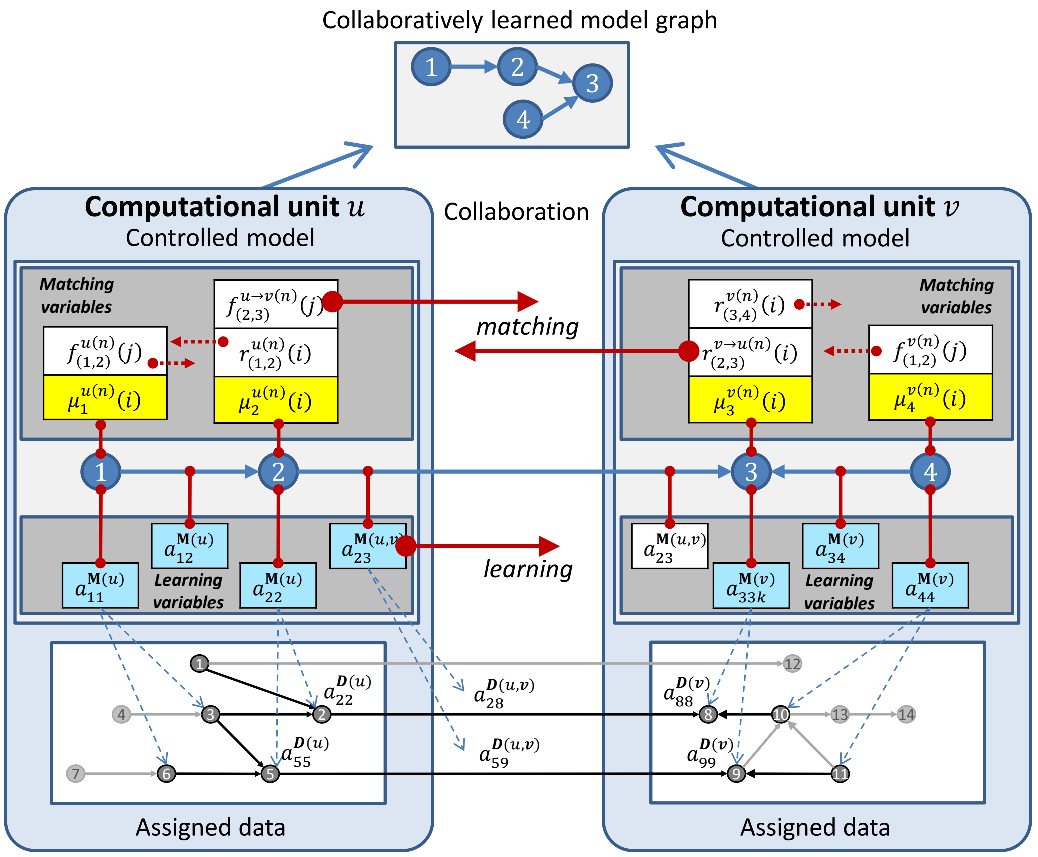


Figure 12: Example of variables and communications in distributed pattern learning algorithm

The only additional communication needed in the pattern learning model is that of attributes of the model links among model graph nodes controlled and learned by different computational resources. But this communication can be avoided if the receiving agent v computes the attribute as follows:

$$a_{km}^{\mathbf{M}(u,v)} = \sum_{i \in V^{\mathbf{D}(u)}, j \in V^{\mathbf{D}(v)}} a_{ij}^{\mathbf{D}(u,v)} \cdot e^{\underbrace{r_{(k,m)}^{v\to u(n)}(i)}_{computed\ by\ unit\ v} + \underbrace{f_{(k,m)}^{u\to v(n)}(j)}_{received\ from\ unit\ u}}$$

While this alternative insignificantly increases the computations performed at agent nodes, and creates situation of duplicate attributes (model links between agents), it will reduce communication between agents. Note, however, that the total run time, and consequently amount of communicated information between the agents, is higher in pattern learning applications due to larger number of iterations needed for convergence.

3.4.4 Learning Frequent Patterns

Learning multiple frequent subgraph patterns in the data may occur naturally: the final model graph learned collaboratively by a group of units will include multiple disconnected components. Each component could be declared a pattern and reported to the users. In general, this is not guaranteed by the model described above. Figure 13a depicts an example where the same "local" pattern at one unit is connected differently to another unit. Learning one unified pattern will result in ambiguity of the actual structure. Instead, we need to enable agents to generate multiple hypotheses of connections between substructures learned at different agents. This can be done by collapsing the pattern instances learned locally by each unit into a single node, and then performing information propagation of pattern labels among the units. This will create a "multi-stage" learning process, where the computational units first learn the subgraphs in their local data, aggregate their instances, and then learn the higher-level "compositional" patterns of connections between such structures (Figure 13b).

One of the crucial steps in both centralized and distributed pattern learning is the initialization. During this step the computational units must decide about the following:

(a) How many model nodes are possible for them to control?

(b) What are feasible links between these nodes?

(c) How to initialize node and link attributes?

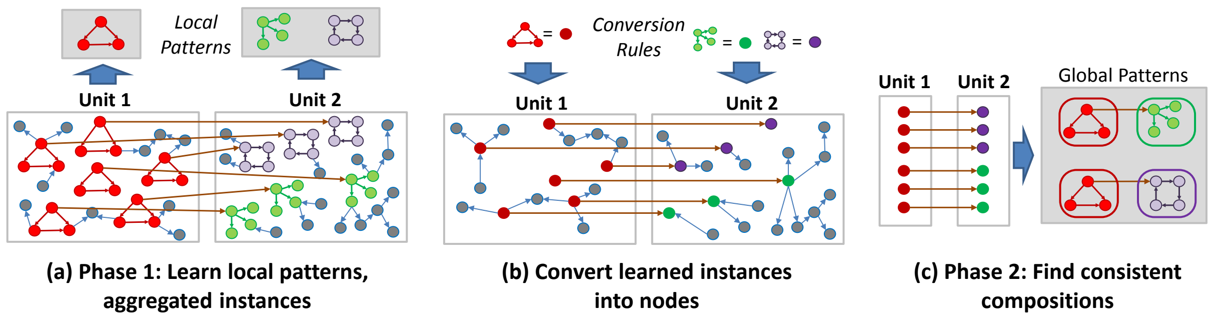

Local Patterns
Unit 1
Unit 2
(a) Phase 1: Learn local patterns, aggregated instances
Conversion Rules
=
=
=
Unit 1
Unit 2
(b) Convert learned instances into nodes
Unit 1
Unit 2
Global Patterns
(c) Phase 2: Find consistent compositions

Figure 13: Learning patterns in distributed manner

To solve these problems, we proceed in the following steps. First, we generate *summary attributes* at a node that describe the neighborhood of the node. Second, we aggregate the nodes into groups using standard clustering or classification algorithms (such as mixture of Gaussians) over summary attributes. Next, we aggregate the nodes in the same class (or cluster), and extract frequent topological subgraphs that are then used to initialize multiple patterns and their corresponding attributes and node-to-node matching probabilities.

3.5 Software Implementation

3.5.1 Software architecture

We started implementation of DSPACE solution by developing a modular architecture, the main components of which are shown in Figure 14:

- **User interface** will allow defining the queries and displaying the results
- **Query processor** segments the query and assigns the analysis tasks to agents
- **BSP controller** synchronizes the agent's operations by calling the "supersteps" and generating the final analysis results
- **Processing units** are individual agents with access to segments of the data, who execute belief estimation computations and collaborate by generating, sending, and incorporating belief messages
- **Messaging system** supports delivery of the messages between the agents in any environment

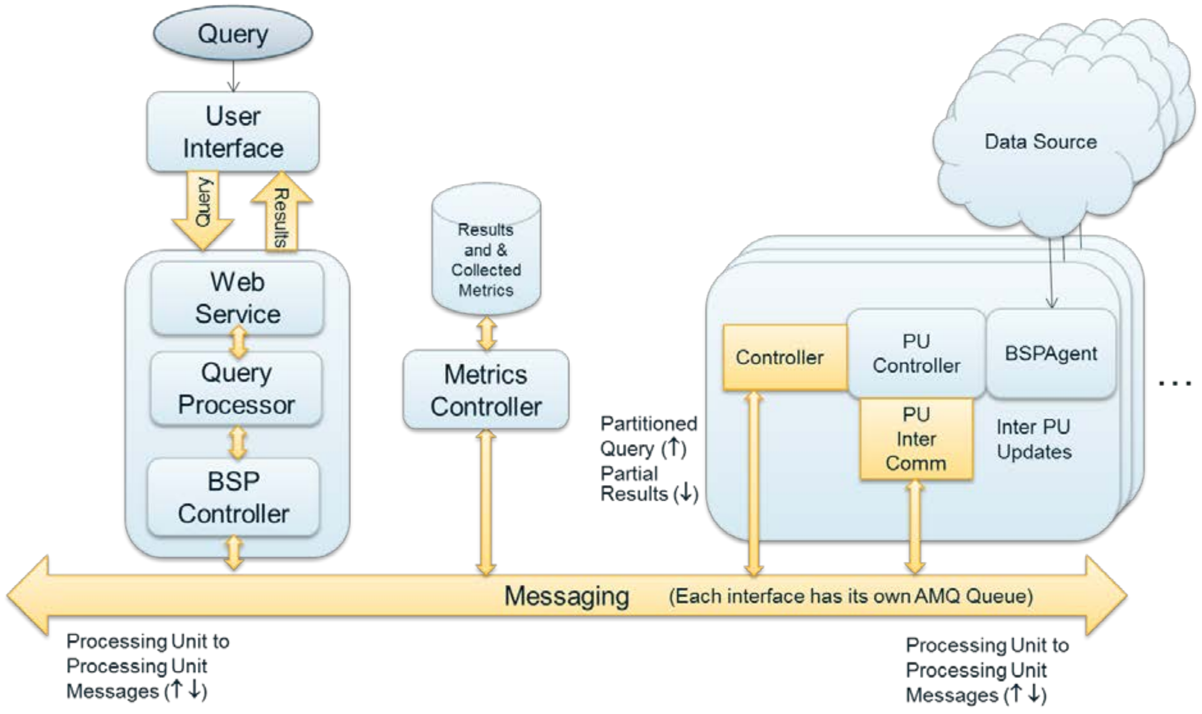
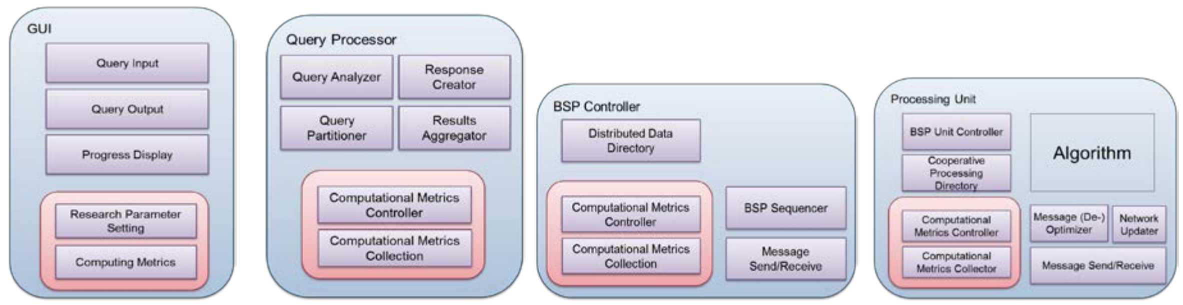


Figure 14: SW architecture of DSPACE system

We conducted the tests of several messaging frameworks, and concluding that Java Messaging Service ActiveMQ implementation together with Protocol Buffers provide the functionality needed for collaborative data analysis. Protocol Buffers are a way of encoding structured data in an efficient yet extensible format. Google uses Protocol Buffers for almost all of its internal RPC protocols and file formats.

Each of the main components contained a set of sub-components, depicted in Figure 15 and described in more detail in the following sections.



(a) User Interface (UI) **(b) Query Processor (QP)** **(c) BSP Controller (BSP)** **(d) Processing Unit (PU)**

Figure 15: Subcomponents of the DSPACE system

3.5.2 Query processor

This component executes the following functions:

- Analyze the query
- Partition the query into the subsets of the EEIs assigned to different processing units (PU) using information about PU's capabilities and accessible data
- Aggregate the results from multiple PUs
- Create a response output and return results to users

The query processor (QP) sends the query partition information and infrastructure and algorithm parameters to the BSP Controller. QP will receive the responses from PUs in the form of belief (probability) messages for the query partitions assigned to PUs.

3.5.3 BSP controller

This component performs the following functions:

- Maintain directory of available processing units and the query parts they can process
- Send assignment of query partitions to the PUs
- Synchronize the distributed data analysis by sending and receive control messages, which include the start and status of the "supersteps"
- Receive computation metrics such as computation time and workload.
- Receive the query results from PUs
- Send all query results to query processor

3.5.4 Processing unit

Processing unit (PU) implements the "agents" – the distributed computation resources that have data access and autonomously collaborate by sending belief messages. PUs implement the following functions:

- Initialize for task
  - Compute mismatch(s)
  - Initialize beliefs
- Loop for super step
  - Update beliefs
  - Generate external messages (/w optimization)
  - Send messages
  - Vote done on super step (pause for coordination)

    - Check for done from controller (if done, exit loop)
    - Receive messages for next super step
    - Search (identify relevant nodes)
- Return results
    - Send results back to controller
    - Send the state of the agent back to controller

The PUs maintain the knowledge of agent organization – i.e., a directory of what other processing units are available, what data they have access to, which ones are connected, and what dependencies are between them. The PUs implement the distributed data analysis algorithm based on Belief Propagation for graph matching. It also includes message optimization (using filtering and compression methods), and the components to compute the metrics of performance (accuracy of results) and process (time and workload logging).

3.5.5 Performance and system metrics

All of the components of DSPACE have the ability to post metrics data to the metrics controller and into the metrics database. When the DSPACE system is processing queries in test mode, it is collecting metrics data using the following steps:

- the DSPACE component calls the logger with an event type and a message. Examples of event types are Query_Submitted, Query_Returned, QueryPartitioner_Started, QueryPartitioner_Completed).
- the logger gets the log request and adds some context information, such as where the log message is coming from, the time of the log message and other identifying information such as the query id that is being processed.
- If the log message is remotely distributed from the metrics controller, the message is sent to the messaging system which transports the message to the metrics controller. The message passing is done using the same message passing system that is used by the query processor to transport messages between processing units and the other controllers (ActiveMQ).
- the metrics controller receives a log message (either from a local component or a remote component) and commits it to the metrics database.

At any time, a user can query the metrics data and generate charts of DSPACE performance. The metrics collection components are extensible and additional test and experiment metrics can be collected and stored in the metrics database. Currently, timing data is being collected for query runs of different data and model sizes. Communication size metrics are also being collected. The types of charts that can be generated are also extensible, currently charts of query execution times for different data and model sizes have been implemented (these can be re-run at any time with new or additional run data to generate new charts) and charts of communication times have been implemented.

3.5.6 User interfaces

We started implementation of DSPACE's user interfaces (UIs) by identifying the list of actions that users must be allowed to perform with the tool, including:

- Input the query/pattern by selecting a file containing the corresponding pattern, and designing the pattern in a graphical form

- Select the data
- Generate the data for experiments (randomly, or from a given dataset)
- Configure the research parameters (define parameters such as the number of agents, configure the analysis algorithms, specify the experiments to be conducted, etc.)
- Show progress of the query (in terms of the time remaining, interactions between distributed collaborating resources, and intermediate results statistics)
- Display intermediate algorithm state (belief-based heatmap over data network)
- Display the results
- Collect status and analysis metrics

Accordingly, DSPACE's UIs included the following GUI elements:

- Dataset and Query graph editing
- Start/stop buttons
- Progress status displays
- Query results display
- Measures display

The interface components for user's interaction with DSPACE system were designed using web services. We implemented three user interfaces:

(1) **Engineering interface** for the scalability experiments. This is a web-based interface that used cytoscape.js (a Javascript version of the tool so it can run in a browser) to visualize the network (graph) data and included the set of parameters that the experiment designer may change (Figure 16)
(2) **Triple-store data connection interface** for experiments with Lehigh University Benchmark (LUBM) dataset (Figure 17)
(3) **The metrics interface** to visualize performance of the system. The metrics user interface is web based similar to the query test interface (Figure 18).

Figure 18 shows two different graphs of query execution times for a variety of data and model sizes (this information was not collected under controlled conditions, therefore, although it is actual DSPACE performance data, it does not reflect meaningful performance results of DSPACE). The graphs are generated from data extracted from the metrics database and visualized using the Google Chart library in the browser. The top bubble chart shows the size of the data set (number of nodes) along the x-axis and the size (number of nodes) in the query along the y-axis. The size of the bubble indicates the execution time to search for a pattern of the model size in a data set of the data size. The color of the bubble, from red, a shorter time, to blue a longer time is also indicated. The second chart shows similar results with each colored line representing a different model size (number of nodes in the query). Note in this graph, if there was no data for that data/model size, the result is shows as zero (such as for a data set size of 2000, query execution time is only shown for a 3 node model query). In this chart, all of the runs with the same data set size and model size are averaged together to determine the value to display for that data and model size line point

These user interfaces communicate with the query controller components via websockets, which allows 2-way communication needed for sending the query and receiving a response (Figure 19).

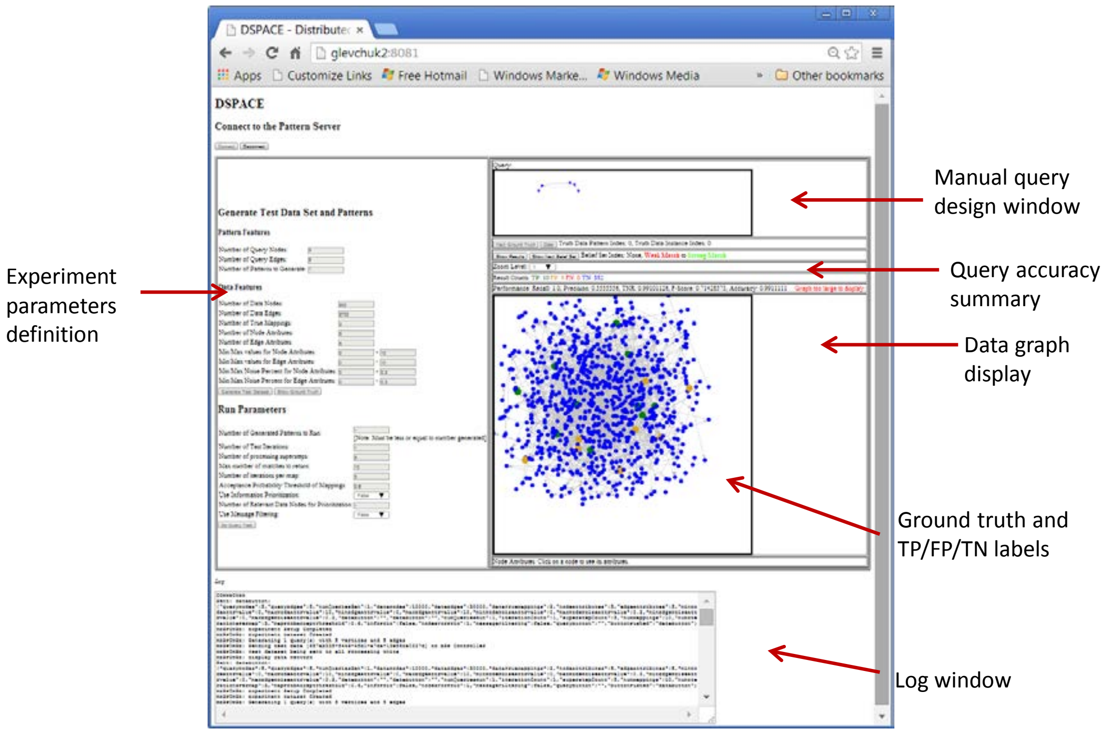


Figure 16: Engineering interface for experiments with synthetic data

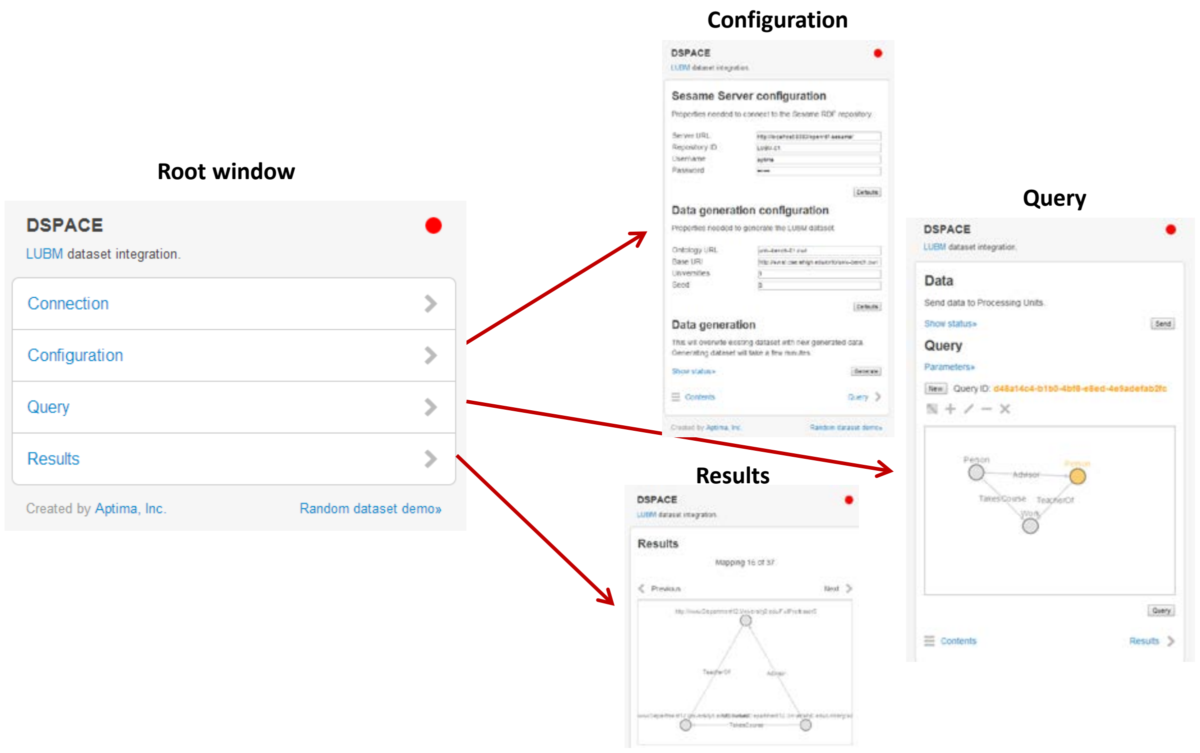
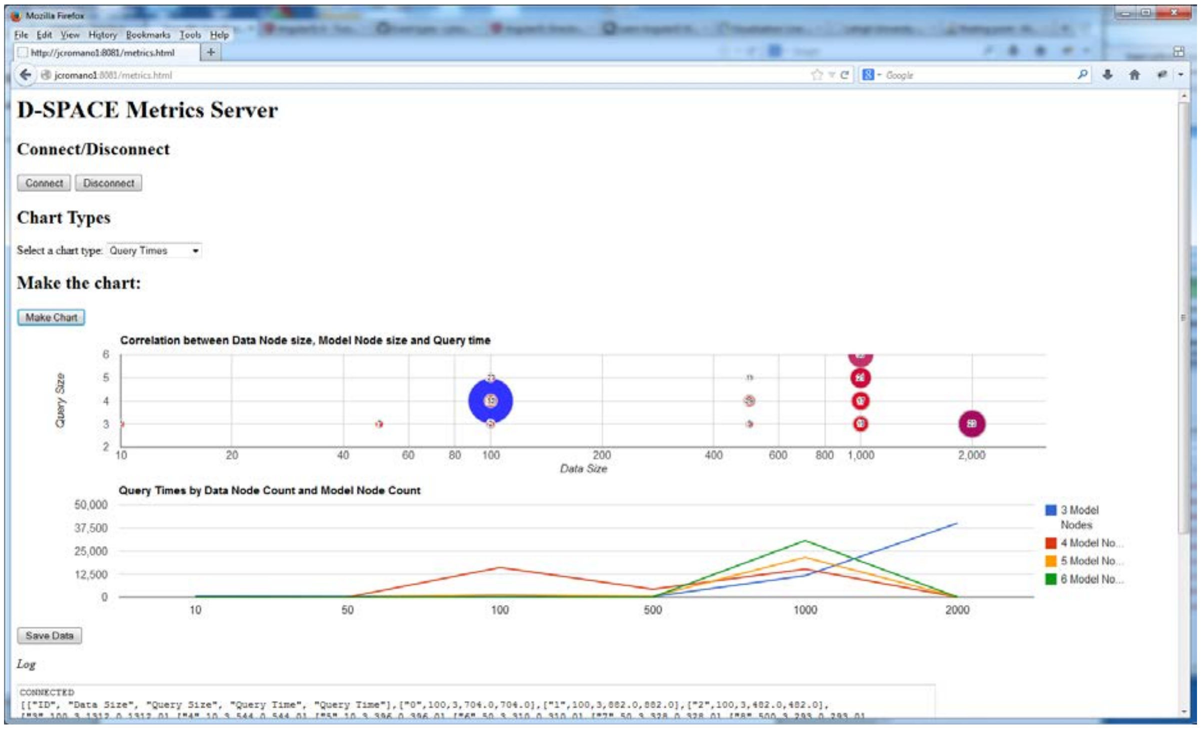


Figure 17: Interface for connecting with LUBM triple store


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://jcromano1:8081/metrics.html
jcromano1:8081/metrics.html

D-SPACE Metrics Server

Connect/Disconnect

Chart Types

Select a chart type: Query Times

Make the chart:

Make Chart
Correlation between Data Node size, Model Node size and Query time
Query Size
Data Size
Query Times by Data Node Count and Model Node Count
3 Model Nodes
4 Model No...
5 Model No...
6 Model No...
Save Data

Log

```
CONNECTED
[["ID", "Data Size", "Query Size", "Query Time", "Query Time"],["0",100,3,704.0,704.0],["1",100,3,882.0,882.0],["2",100,3,482.0,482.0],
["3",100,3,1312.0,1312.0],["4",10,3,544.0,544.0],["5",10,3,396.0,396.0],["6",50,3,310.0,310.0],["7",50,3,328.0,328.0],["8",500,3,293.0,293.0]
```

Figure 18: Metrics interfaces

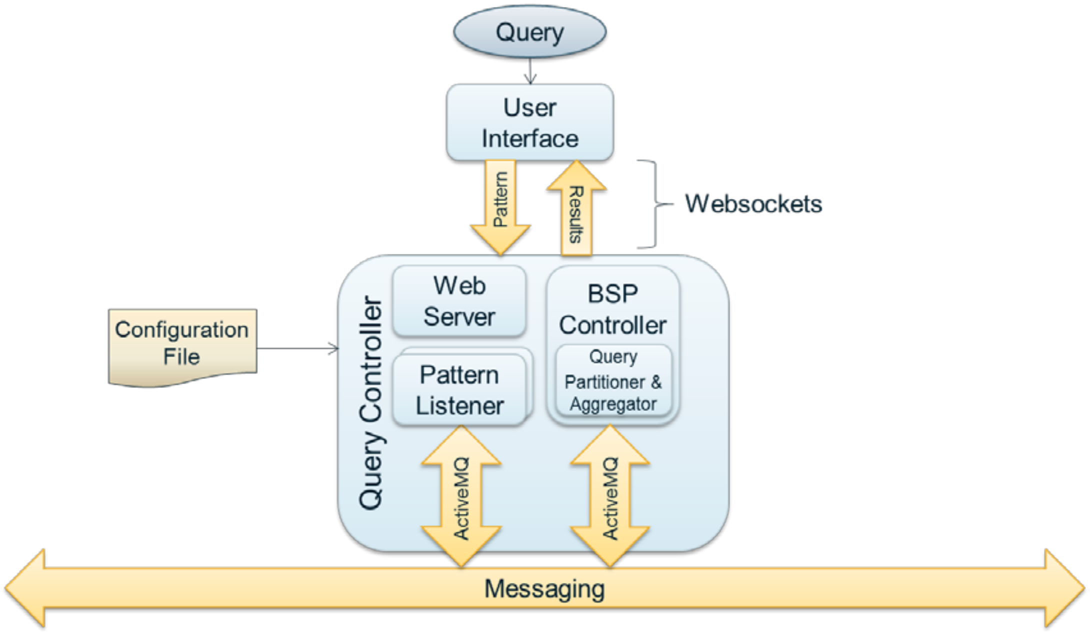


Figure 19: Interactions between user interfaces, controller components, and the agents

4. RESULTS AND DISCUSSION

We conducted experiments with synthetic data, both randomly generated one and the LUBM benchmarking data that was designed as realistic triple-store datasets.

4.1 Experiments with randomly generated data

In the first set of experiments, we used randomly generated query and data graphs to validate three hypotheses. First, we showed that computational complexity of our querying solution is linear in the size of the data, and that our solution is robust to data noise. Second, we showed that distributed solution produces comparable accuracy to the centralized graph matching, while providing further computational improvements on distributed peer-to-peer networks. Finally, we showed that scalability improvements using prioritization and belief message compression do provide reduction in the local computations by units and communication among them.

To achieve these goals, we used the synthetic graph generator jgrapht[2]. We generated a set of queries, their instances, and the data graph. Then the instances, - the true matches to the query that we want the algorithm to retrieve, - were embedded in the data graph, and the noise added to the values (Figure 20). Using synthetic graph data allowed us to vary the size of the queries and data graphs, number of true query matches in data, the amount of noise in attributes, graph density and attribute value range, etc. Availability of ground truth allowed scoring the accuracy of our solution.

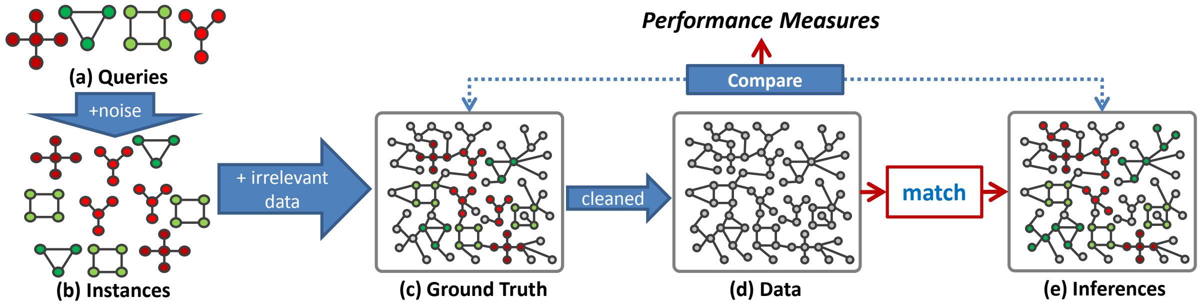


Figure 20: Experimental setup

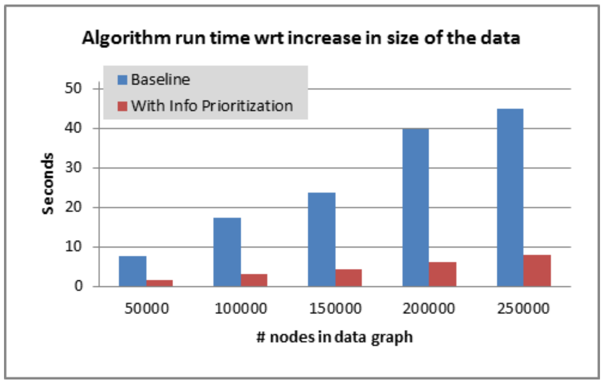
4.1.1 Solution scalability

We assessed our algorithm using the metrics of computational run-time (in seconds) for several stages of the algorithm, including:

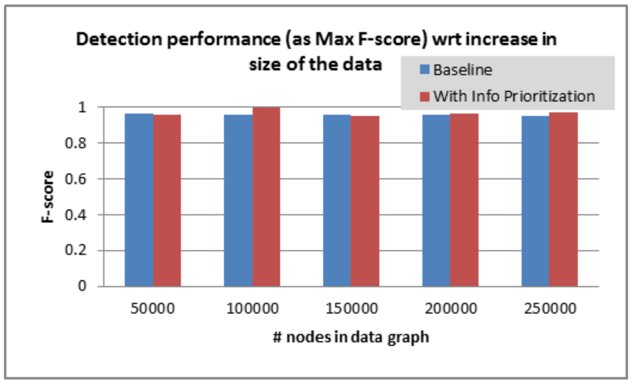
- **Map-log update**: this is the time to update belief messages $\mu_k(i)$ in Eq. (5)
- **Message generation**: this is the time to generate forward and backward messages $f_{(m,k)}(j)$ and $r_{(m,k)}(i)$ in Eq. (6)-(7)

To evaluate impact of the communication independent of the actual network, we computed the **communication workload** as number of external message values that were generated and needed to be communicated between the computational units. Finally, we computed efficiency scores including **recall**, **precision**, and **f-score**.

[2] www.jgrapht.org



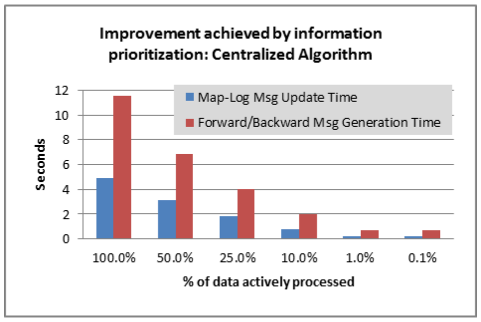
(a) Information prioritization (10%) provides order of magnitude improvement in run time



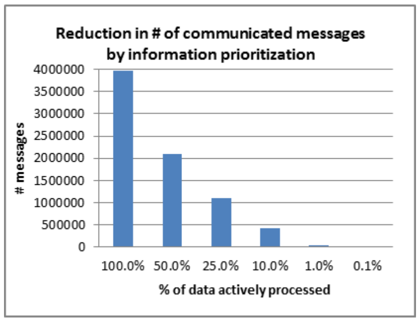
(b) Retrieval accuracy is comparable for different sizes of the data graph

Figure 21: Results of scalability experiments: algorithm run time is linear with size of data

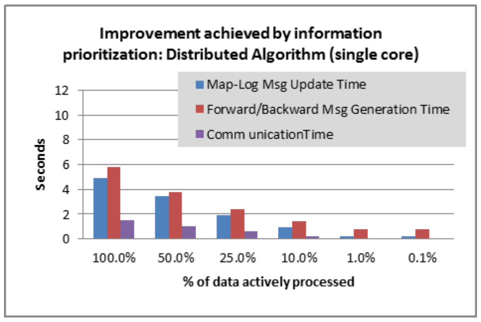
Figure 21 shows that our matching-based data querying solution has run-time complexity linear with the size of the data. The information prioritization (Figure 15 shows results for 10% of data nodes actively processed at every iteration) provides order of magnitude improvement in the run time, computed as the sum of belief message updates from Eq. (5)-(7) in the centralized case (Figure 21a), at the expense of some reduction in the retrieval accuracy (measured by F-score; Figure 21b).



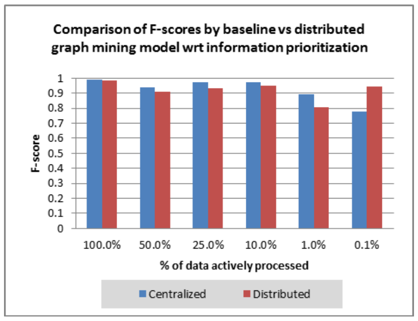
(a) Effect of information prioritization on centralized implementation. # data nodes = 100K



(c) Information prioritization also decreases the number of communicated messages



(b) Effect of information prioritization on distributed implementation ran on a single machine with minimal communication delays



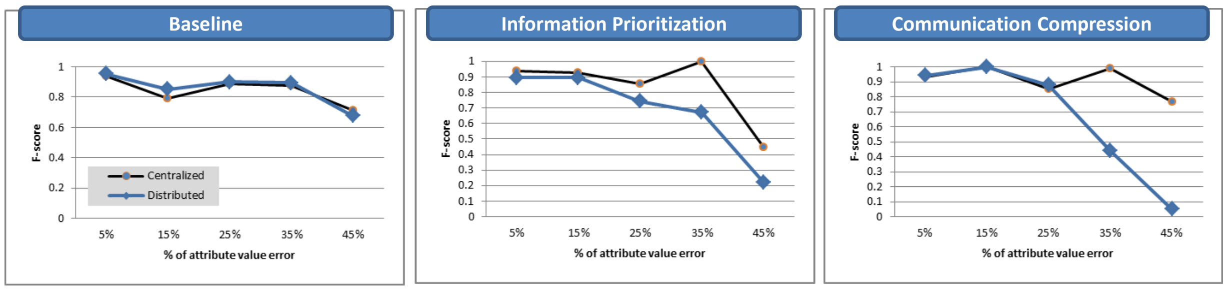
(d) Accuracy of centralized and distributed implementations is similar for different levels of information prioritization

Figure 22: Improvements in scalability can be achieved by using information prioritization model which reduces the amount of data actively processed at each iteration

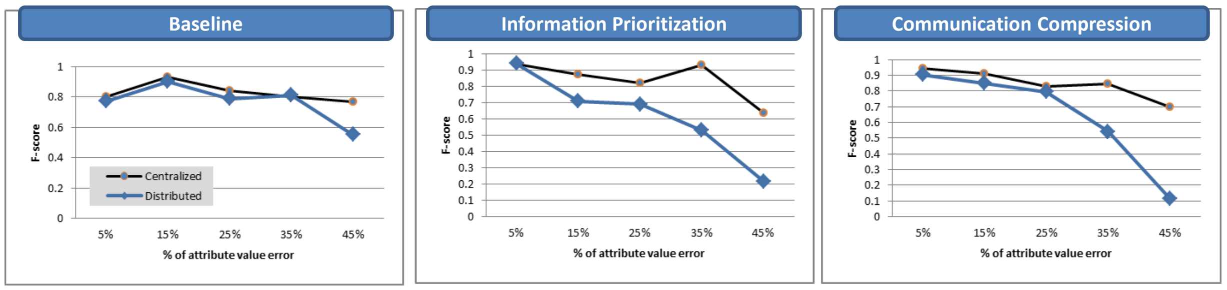
The performance of the algorithm can be tuned, trading-off the accuracy with computational complexity. An order of magnitude reduction in update time can be achieved if the number of “relevant” nodes processed at the computational units is 10% of the total number of data nodes (Figure 22a), while further reduction in the size of actively processed data nodes provides only marginal reduction in computation time. Decentralized implementation had a run-time on the same total computational resources slightly better than centralized implementation (Figure 22b), which was due to minimal communication delays. However, the information prioritization drastically reduces the number of messages that must be communicated between the units (Figure 22c), hence the distributed implementation promises to have linear-scale improvement (in terms of the number of compute units). The accuracy of distributed implementation was comparable to centralized implementation (Figure 22d) when the attributes did not contain any noise, which is essentially a case on attributed graph isomorphism and is the type of solution provided by standard querying engines such as SPARQL.

4.1.2 Solution sensitivity to noise

Figure 23 shows how the accuracy of our algorithm is affected by the noise in the data. While the baseline algorithm is robust to noise, the performance of the algorithms with scalability configurations (“information prioritization” or “communication compression”) degrades as the range of error in attribute values is increasing. In particular, the degradation is more pronounced in the distributed case, since the local decisions by computational units of which data nodes are more relevant and which forward/backward messages must be communicated are becoming less accurate globally with increase in data noise.



(a) Results of retrieval with data that contains 1 true match to a query



(b) Results of retrieval with data that contains 10 true matches to a query

Figure 23: Results of noise sensitivity experiments

4.2 Experiments with LUBM data

LUBM is an artificial data generator of graph data that represents entities and interactions of universities, staff, students, classes, publishing (Figure 24). LUBM graph is fully connected, containing the knowledge fragments that describe the relations of and interactions between entities (Figure 25).

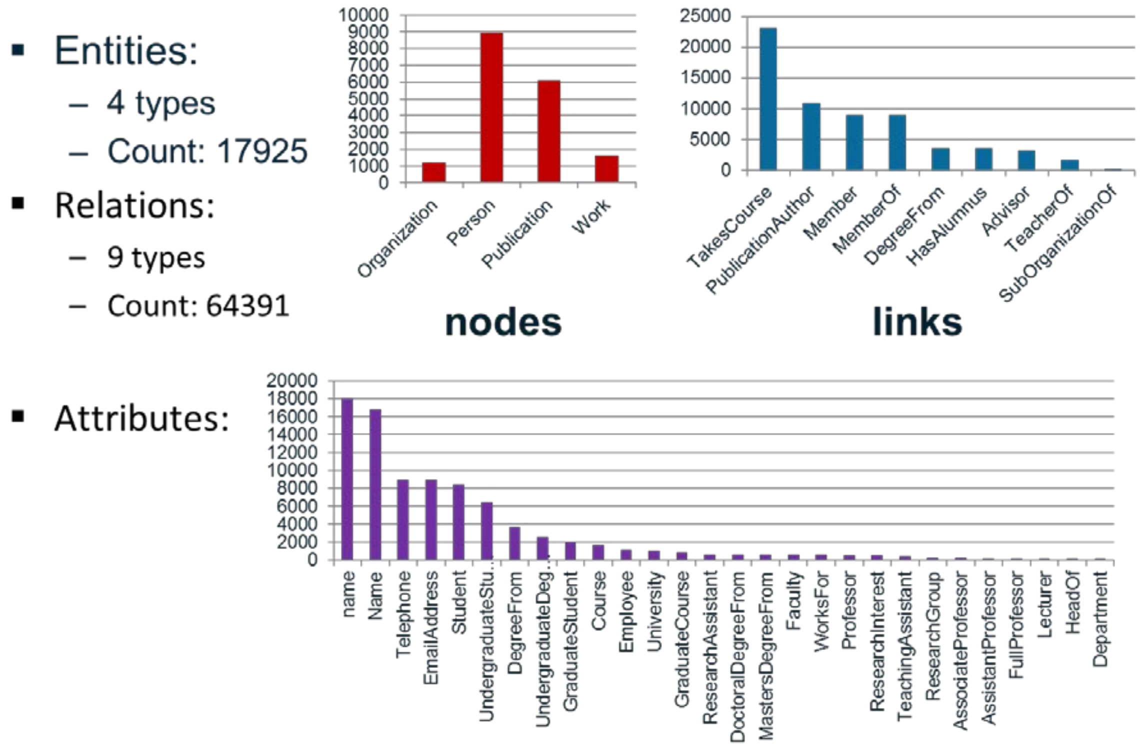


Figure 24: Example statistic of a single-university LUBM data subset

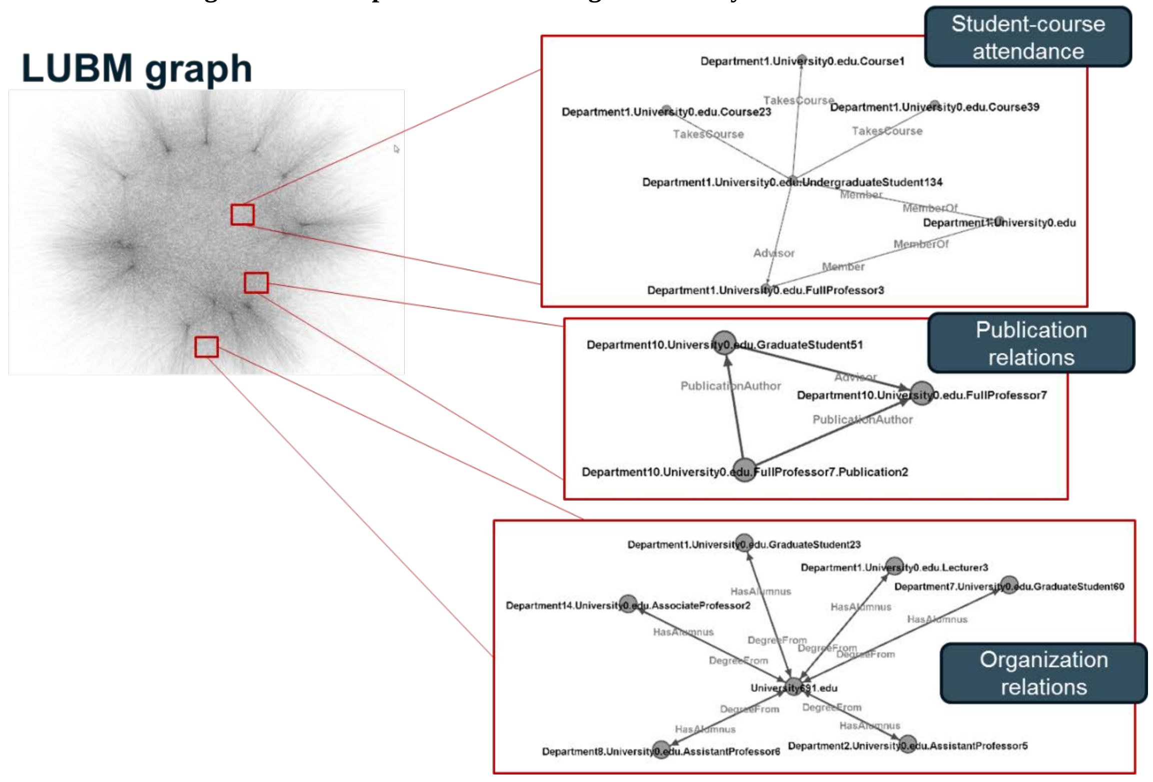

LUBM graph
Student-course attendance
Department1.University0.edu.Course1
Department1.University0.edu.Course23
Department1.University0.edu.Course39
TakesCourse
Department1.University0.edu.UndergraduateStudent134
Member
MemberOf
Department1.University0.edu
Advisor
Department1.University0.edu.FullProfessor3
Publication relations
Department10.University0.edu.GraduateStudent51
PublicationAuthor
Advisor
Department10.University0.edu.FullProfessor7
Department10.University0.edu.FullProfessor7.Publication2
Organization relations
Department1.University0.edu.GraduateStudent23
Department1.University0.edu.Lecturer3
Department7.University0.edu.GraduateStudent60
Department14.University0.edu.AssociateProfessor2
HasAlumnus
DegreeFrom
University631.edu
Department8.University0.edu.AssistantProfessor6
Department2.University0.edu.AssistantProfessor5

Figure 25: Knowledge fragments are subgraphs in LUBM data graph

We used DSPACE to analyze the frequency of knowledge patterns in LUBM data. We extracted the knowledge subgraphs by selecting key nodes and the 1-2 hop neighborhood around them, treated these fragments as queries, and then tried to find other knowledge fragments that matched the query. Some of the knowledge patterns occurred frequently in the data, while others do not (Figure 26). Several examples of these queries and the matches are depicted in Figures 27-29.

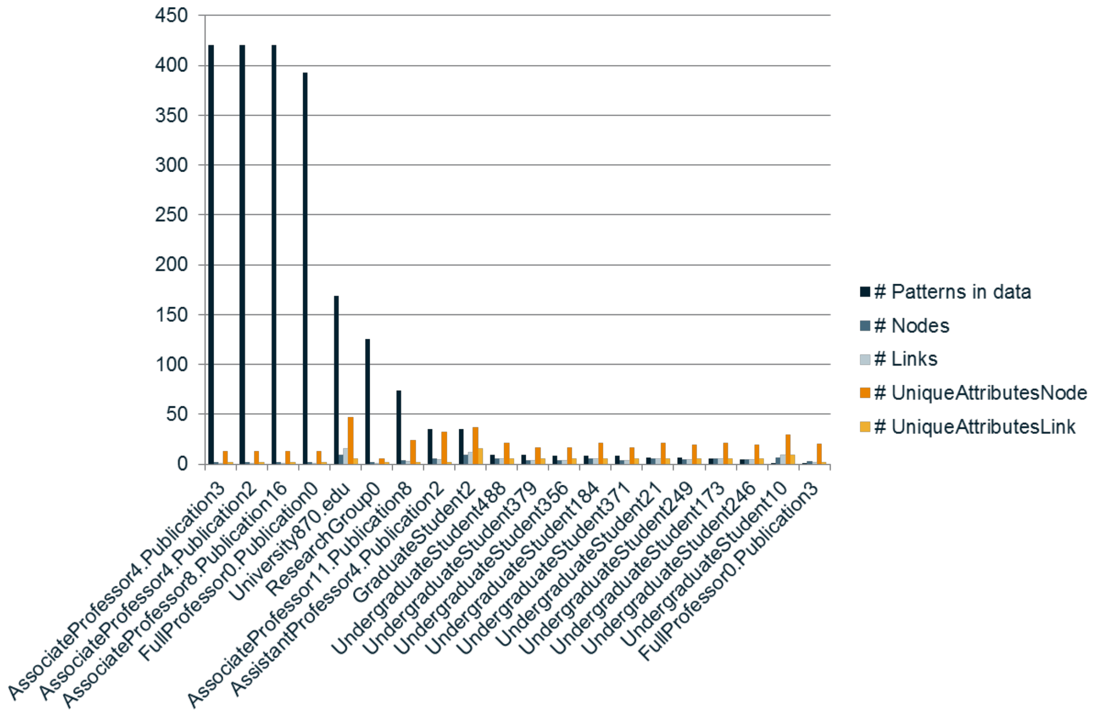
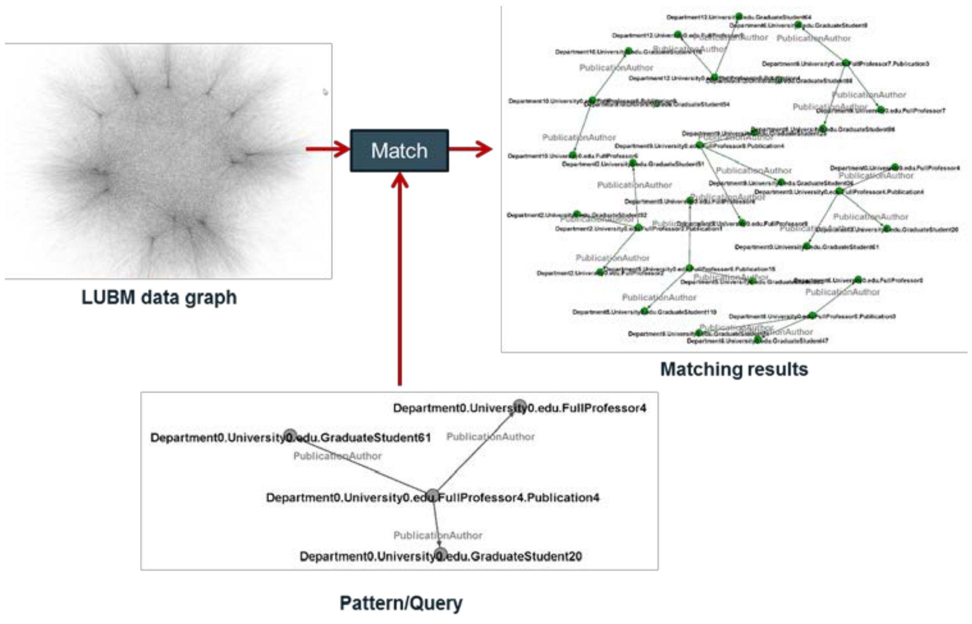


Figure 26: Knowledge fragments are subgraphs in LUBM data graph



LUBM data graph
Match
Matching results
Department0.University0.edu.FullProfessor4
Department0.University0.edu.GraduateStudent61
PublicationAuthor
PublicationAuthor
Department0.University0.edu.FullProfessor4.Publication4
PublicationAuthor
Department0.University0.edu.GraduateStudent20
Pattern/Query

Figure 27: Examples of queries and matches in LUBM data – Publication pattern

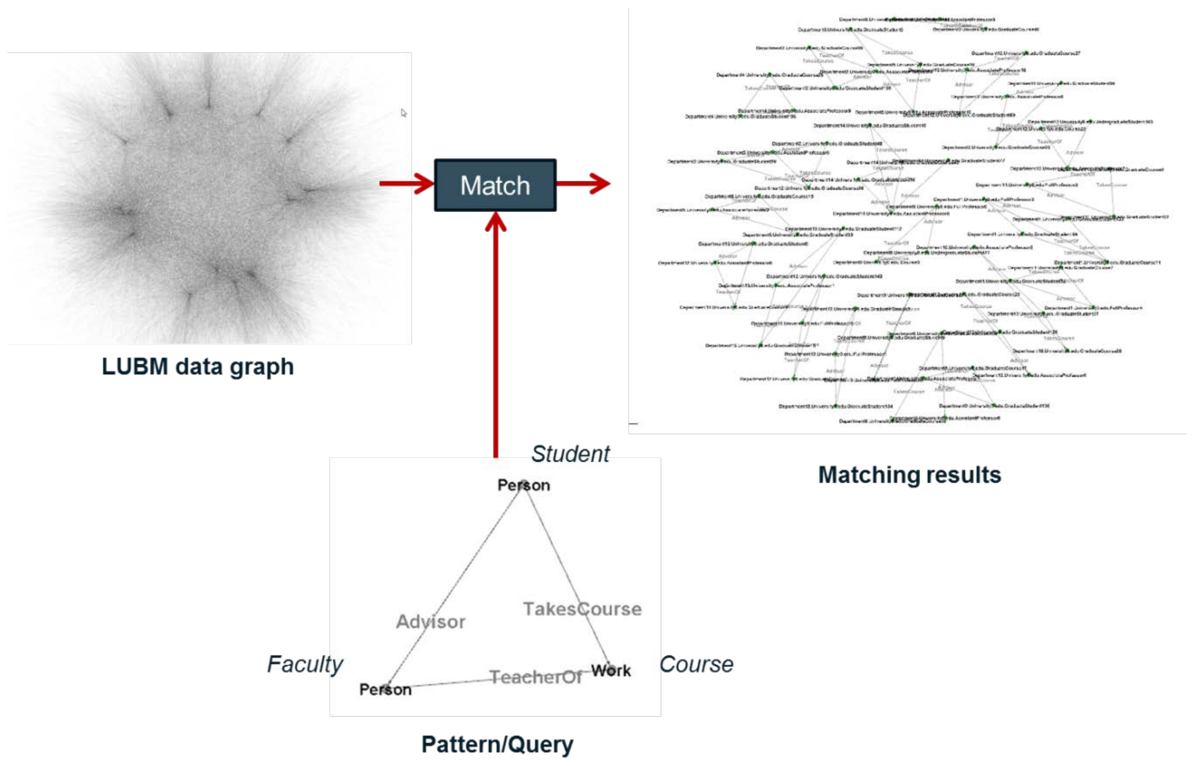


Figure 28: Examples of queries and matches in LUBM data – Student-adviser pattern

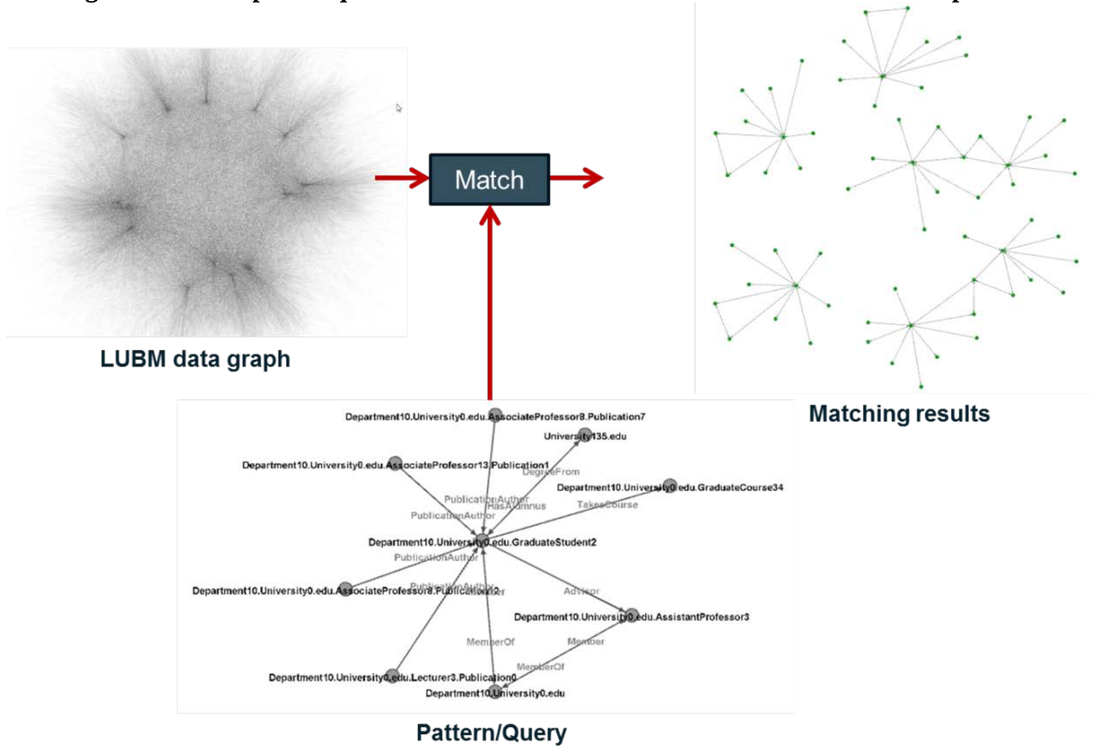

Match
LUBM data graph
Matching results
Department10.University0.edu.AssociateProfessor8.Publication7
University135.edu
Department10.University0.edu.AssociateProfessor13.Publication1
DegreeFrom
Department10.University0.edu.GraduateCourse34
PublicationAuthor
HasAlumnus
TakesCourse
PublicationAuthor
Department10.University0.edu.GraduateStudent2
PublicationAuthor
Department10.University0.edu.AssociateProfessor8.Publication12
PublicationAuthor
Advisor
Department10.University0.edu.AssistantProfessor3
MemberOf
Member
MemberOf
Department10.University0.edu.Lecturer3.Publication0
Department10.University0.edu
Pattern/Query

Figure 29: Examples of queries and matches in LUBM data – Student attendance pattern

5. CONCLUSION

In this project we developed a DSPACE system for performing distributed queries against large-scale relational (graph) datasets which cannot employ Cloud distributed file sharing and processing systems. The solution to this problem is an autonomous peer-to-peer network of computing units that can perform data exploration tasks, such as retrieving information based on complex ambiguous queries, or discovery of frequent graph patterns in the data. We implemented a data exploitation model as a distributed collaborative inexact graph matching, which defines a collaborative data processing policy of local computations at and communication between processing units. We showed that baseline distributed implementation has the same accuracy as the centralized solution. Both centralized and distributed algorithms can achieve improvement in scalability using priority-based filtering and message compression techniques, at the expense of some degradation in the retrieval accuracy.

Our current research is focused on improvements to the distributed collaborative graph analysis algorithms, including distributed graph pattern learning, adaptive prioritization and filtering, and graph indexing, to provide further scalability and accuracy improvements.

6. REFERENCES

Aggarwal, C., Khan, A., Yan, X. (2011) On Flow Authority Discovery in Social Networks. SIAM Conference on Data Mining (SDM), pp, 522–533.

Anker, T., D. Dolev, and B. Hodd (2008) Belief Propagation in Wireless Sensor Networks - A Practical Approach, Proceedings of the 3rd International Conference on Wireless Algorithms, Systems, and Applications, 466 – 479.

Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., & Sudarshan, S. (2002). Keyword searching and browsing in databases using BANKS. In Data Engineering, 2002. Proceedings. 18th International Conference on (pp. 431-440). IEEE.

Brocheler, M., A. Pugliese, V. P. Bucci, and V. S. Subrahmanian (2010) COSI: Cloud oriented subgraph identification in massive social networks, ASONAM, pp. 248-255.

Bryant, M., Johnson, P., Kent, B. M., Nowak, M., & Rogers, S. (2008). Layered sensing: its definition, attributes, and guiding principles for afrl strategic technology development. Wright-Patterson Air Force Base, Ohio: Sensors Directorate, Air Force Research Laboratory.

Chechetka, A., and C. Guestrin (2010) Focused Belief Propagation for Query-Specific Inference. In Artificial Intelligence and Statistics (AISTATS).

Chen, W., Levy, R., & Decker, K. (2007) An Integrated Multi-Agent Coordination Including Planning, Scheduling, and Task Execution. Proceedings of the Workshop of Coordinating Agent's Planning and Scheduling (CAPS) at the Sixth AAMAS, Hawaii, USA.

Crick, C., A. Pfeffer (2003) Loopy Belief Propagation as a Basis for Communication in Sensor Networks, UAI, pp. 159-166.

Dahm, W. (2010, 15 May). Technology Horizons: A Vision for Air Force Science and Technology During 2010-2030. Vol. 1, AF/ST-TR-10-01-PR.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal statistical Society, 39(1), 1-38.

Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J. & Hong, W. (2005). Model-based Approximate Querying in Sensor Networks. International Journal on Very Large Data Bases (VLDB), 14(4), 417-443.

Ding, C., Song, B., Morye, A., Farrell, J. A., & Roy-Chowdhury, A. K. (2012). Collaborative sensing in a distributed PTZ camera network. Image Processing, IEEE Transactions on, 21(7), 3282-3295.

Elidan, G., I. Mcgraw, and D. Koller (2006) Residual belief propagation: Informed scheduling for asynchronous message passing, Proceedings of the Twenty-second Conference on Uncertainty in AI (UAI), Boston, MA.

Gonzalez, J., Low, Y., and Guestrin C. (2009) Residual Splash for Optimally Parallelizing Belief Propagation. In Artificial Intelligence and Statistics (AISTATS).

He, H., Wang, H., Yang, J., and Yu, P. S. (2007, June). BLINKS: ranked keyword searches on graphs. In Proceedings of the 2007 ACM SIGMOD international conference on Management of data (pp. 305-316). ACM.

Khan, A., N. Li, Z. Guan, X. Yan, S. Chakraborty, and S. Tao (2011, June) Neighborhood Based Fast Graph Search in Large Networks, SIGMOD'11.

Levchuk, G., & Pattipati, K. (2013). Design of Distributed Command and Control for Collaborative Situation Assessment, Proceedings of ICCRTS 2013.

Levchuk, G., J. Roberts, and J. Freeman (2012) Learning and Detecting Patterns in Multi-Attributed Network Data, AAAI Fall Symposium on Social Networks and Social Contagion.

Levchuk, G., C. Shabarekh, and C. Furjanic (2011) Wide-threat Detection: Recognition of Adversarial Missions and Activity Patterns in Empire Challenge 2009, Proceedings of SPIE Defense and Security Symposium, Orlando, FL.

Madden, S., Franklin, M. J., Hellerstein, J. M., & Hong, W. (2003, June). The design of an acquisitional query processor for sensor networks. In Proceedings of the 2003 ACM SIGMOD international conference on Management of data (pp. 491-502). ACM.

Malewicz, G., M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, and G. Czajkowski (2010) Pregel: a system for large-scale graph processing, In SIGMOD, pages 135-146.

Mathew, G., Surana, A., & Mezic, I. (2010). Uniform Coverage Control of Mobile Sensor Networks for Dynamic Target Detection. In Proceedings of the 49th IEEE Conference on Decision and Control, Atlanta, 2010.

Pfeffer, A., T. Tai (2012) Asynchronous Dynamic Bayesian Networks, CoRR, abs/1207.1398.

Rohloff, K., & Schantz, R. E. (2010, October). High-performance, massively scalable distributed systems using the MapReduce software framework: the SHARD triple-store. In Programming Support Innovations for Emerging Distributed Applications (p. 4). ACM.

Rohloff, K., & Schantz, R. E. (2011, June). Clause-iteration with MapReduce to scalably query datagraphs in the SHARD graph-store. In Proceedings of the fourth international workshop on Data-intensive distributed computing (pp. 35-44). ACM.

Singhal, A. (May 16, 2012). Introducing the Knowledge Graph: Things, Not Strings. Official Blog (of Google). Retrieved April, 2014

Sutton, C., and A. McCallum (2007) Improved Dynamic Schedules for Belief Propagation, Conference on Uncertainty in Artificial Intelligence (UAI)

Tran, T., Wang, H., Rudolph, S., & Cimiano, P. (2009, March). Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on (pp. 405-416). IEEE.

Wu, Y., Yang, S., Srivatsa, M., Iyengar, A., & Yan, X. (2013). Summarizing answer graphs induced by keyword queries. Proceedings of the VLDB Endowment, 6(14), 1774-1785.

Yang, S., Yan, X., Zong, B., & Khan, A. (2012, May). Towards effective partition management for large graphs. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (pp. 517-528). ACM.

Yao, Y. & Gehrke, J. (2003) Query processing in sensor networks. Conference on Innovative Data Systems Research (CIDR).

Yedidia, J. S., Freeman, W. T., & Weiss, Y. (2000, December). Generalized belief propagation. In NIPS (Vol. 13, pp. 689-695).

Zhang, L., Tran, T., & Rettinger, A. (2013). Probabilistic query rewriting for efficient and effective keyword search on graph data. Proceedings of the VLDB Endowment, 6(14), 1642-1653.

Rogers, A., Farinelli, A., Stranders, R., & Jennings, N. R. (2011). Bounded approximate decentralised coordination via the max-sum algorithm. Artificial Intelligence, 175(2), 730-759.

LIST OF ACRONYMS

ACRONYM	**DESCRIPTION**
AFRL	Air Force Research Laboratory
SW	Software
JMS	Java Message Service
API	Application Programming Interface
BSP	Bulk Synchronous Parallel
ISR	Intelligence Surveillance and Reconnaissance
BAA	Broad Agency Announcement
SQL	Structured Query Language
SPARQL	SPARQL Protocol and RDF Query Language
RDF	Resource Description Framework
BP	Belief Propagation
SLBP	Smoothed Loopy Belief Propagation
LIDAR	Light Detection And Ranging
GMTI	Ground Moving Target Indicator
EM	Expectation Maximization
DSPACE	Distributed Sensing and Processing Adaptive Collaborative Environment
LUBM	Lehigh University Benchmarking